# Application of Finite Automata in Debugging Natural Language Vocabularies

Tomasz Kowaltowski, Cláudio L. Lucchesi and Jorge Stolfi*
Department of Computer Science
University of Caminas
13081-970 Campinas, SP, Brazil
{tomasz,lucchesi,stolfi}@dcc.unicamp.br

**Abstract**

Finite acyclic automata can be used as a very versatile tool in many applications involving natural language vocabularies. This work describes some experiments in "debugging" semi-automatically such vocabularies, i. e. suggesting non-existent and missing words. Partial statistics are shown for Portuguese, Italian and English vocabularies.

## 1 Introduction

Finite automata provide a convenient tool for representing and manipulating efficiently sets of words. Most traditional applications are found in compiler construction, where such automata are used to model and implement lexical analyzers. Interesting examples of applications of finite automata to solve specific problems in natural language processing can be found, for instance, in a volume edited by Gross and Perrin [2]. However, the idea of compressing a very large vocabulary of words of a natural language into a minimal acyclic deterministic automaton seems to be fairly new. One example is described by Appel and Jacobson [1] in which the automaton is used for the popular game Scrabble. In another paper Lucchesi and Kowaltowski [5] described the use of finite automata as a basis for building efficient spelling checkers and advisers, and mentioned other possible applications. Some of the important properties of finite automata which are explored in all of these applications are the compactness of the representation and the speed with which the information can be retrieved (see [5]). In a companion paper ([4]) we study how to reduce even further the size of the representation without loss of access efficiency.

*Authors are listed in alphabetical order.

In this work we focus on another application of such automata, namely the "debugging" of natural language vocabularies. Such vocabularies are large collections (from tens of thousands up to hundreds of thousands) of words. These collections are typically compiled from many disparate sources, and contain many misspellings and omissions. Detecting such errors is usually a very tedious and time consuming task. It becomes even more complex within the framework of Romance languages, with their large number of forms, particularly in the case of verbs. An account of the compilation of an English vocabulary which forms the basis of the UNIX spelling checker spell is given by McIlroy [6].

In order to address the problem of debugging a vocabulary, we built a set of language-independent tools which help in detecting such errors. Some of these tools identify words which do not seem to conform to any simple pattern, and thus are candidates to be excluded from the vocabulary. Other tools suggest words which are not in the vocabulary but whose addition would simplify in some sense its automaton, and are thus candidates for inclusion in the vocabulary. We have successfully used these tools to debug large Portuguese, Italian and English vocabularies obtained from various sources. It should be stressed that the Portuguese and English vocabularies we had available were reasonably consistent, but this was not the case of the Italian one which was very incomplete. This provided a very interesting example and we were able to more than double the number of its valid words as a result of using our technique.

Since natural languages have many irregularities and exceptions, the debugging process cannot be completely automated. No matter how good a program may be at identifying likely errors, it can only make suggestions, whose validity must eventually be judged by a human expert.

## 2 Programming environment and tools

All the software described here was implemented in the language MODULA-3 [7, 3], 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 and its availability on most UNIX workstations.

### 2.1 Basic library

Our debugging tools are built on top of a library of efficient routines for manipulating large acyclic automata, which we developed in the last few years. The main modules (abstractions) in this library are:

- Dag: implements acyclic directed graphs which are used for the next abstraction;
- Reduced: implements reduced finite acyclic automata; 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;

- ReducedPair: implements efficiently boolean operations between the sets of words accepted by two automata.

The library also includes many auxiliary abstractions for handling texts, files, etc. Altogether, the basic library contains some 6,000 lines of source code.

## 2.2 Tools

The debugging tools are implemented as three separate programs:

- MaintainAutomaton: a simple program (about 600 lines) for building finite automata from wordlists, including or excluding sets of words, collecting statistics, etc;
- AutoAnalysis: the main debugging tool (about 1,000 lines) which performs several analyses on a given automaton, including: *unproductive states* (states which contribute very few words to the language), *similar states* (pairs of states which accept very similar sets of suffixes), *lexical classes* (states which seem to correspond to major lexical categories), *lexical radicals* (sets of prefix strings which seem to correspond to lexical stems);
- IdealClasses: an analysis tool (about 1,300 lines), still under tests, which tries to assign the words of the language to a given set of lexical classes, on the basis of their inflection patterns.

# 3 The AutoAnalysis program

Below we describe AutoAnalysis, sofar our main debugging tool, in more detail. The debugging experiments reported in Section 4 were largely based on its output.

## 3.1 Unproductive states

One of the functions of the AutoAnalysis tool is to look for possible misspellings and "foreign" words in the vocabulary, by looking for states that are used by few words.

Given a state $t$ of the automaton, we donote by $P(t)$ the set of *prefixes of* $t$, i. e. the set of words spelled by all paths through the automaton starting at its initial state and ending at $t$. Analogously, $S(t)$ denotes the set of *suffixes of* $t$, i. e. the set words spelled by all paths starting at $t$ and reaching some final state.

It is easy to see that the set of words in the language whose paths go through a particular state $t$ is $W(t) = P(t) \cdot S(t)$,[1] and that the number of such words is exactly $|P(t)| \cdot |S(t)|$. The AutoAnalysis program considers that a state $t$ is *unproductive* if $|W(t)| \leq p_0$, where $p_0$ is a user-specified parameter (typically 1 to 3). A state is therefore unproductive if removing it

[1]The operator '$\cdot$' is used to denote concatenation of sets of strings.

from the automaton (together with all its incoming and outgoing transitions) would eliminate at most $p_0$ words from the language.

For each unproductive state found, the program prints the list $W(t)$ of words that use it. After merging and sorting these lists, and eliminating duplicate entries, we manually check the validity of these words and remove those that are genuine mistakes. Experience shows that misspellings and "foreign" words present in the vocabulary are very likely to come up in this listing. See Section 4 for examples.

## 3.2 Similar states

Another function of the AutoAnalysis program is to suggest words that may be missing from the vocabulary, by looking for pairs of *similar* states.

Intuitively, two states $t_1$ and $t_2$ are similar if they could be collapsed into a single state by adding a relatively small number of words to the language. The definition of "relatively small" can be controlled by the user to some extent, but basically it means that the prefix sets $P_i = P(t_i)$ of both states must be small, and their suffix sets $S_i = S(t_i)$ must differ by a small number of strings, in relative and/or absolute terms. More precisely, the program considers $t_1$ and $t_2$ to be *similar* if and only if[2]

1. $|S_1 \setminus S_2| \leq p_1$ and $|S_2 \setminus S_1| \leq p_1$
2. $|S_1 \setminus S_2|/|S_1| \leq p_2$ and $|S_2 \setminus S_1|/|S_2| \leq p_2$
3. $|S_1 \cap S_2| \geq p_3$
4. $|P_1| \leq p_4$ and $|P_2| \leq p_4$
5. $|P_1| \cdot |S_2 \setminus S_1| + |P_2| \cdot |S_1 \setminus S_2| \leq p_5$

where $p_1$ through $p_5$ are user-given parameters. For each pair $(t_1, t_2)$ of distinct states that meets these criteria, the program outputs the set of strings

$$U(t_1, t_2) = P_1 \cdot (S_2 \setminus S_1) \cup P_2 \cdot (S_1 \setminus S_2)$$

These are the words whose addition to the vocabulary would cause the two states to collapse into one (assuming that neither state is reachable from the other).

The choice of the parameters $p_1$ through $p_5$ usually depends on the vocabulary, and should maximize the number of valid suggestions, without generating too many invalid ones. At present, we do this by trial and error. (As a matter of fact, some of the criteria above where introduced in response to previous experiments which produced exaggerated number of suggestions.) Usually a few test runs of the AutoAnalysis program are enough to obtain a satisfactory list of suggestions.

[2]The operator '\' is used to denote set difference.

After merging all the output lists $U(t_1, t_2)$, and eliminating repeated entries, we manually check the validity of each suggestion. For most of the vocabularies we have obtained so far, this process typically produces thousands of distinct suggestions, 10 to 50% of them being valid words that are indeed missing from the vocabulary.

Typically, the manual inspection also uncovers a number of misspellings and "foreign" words in the vocabulary, which are revealed by the appearance of peculiar suggestions formed from their prefixes and suffixes.

After adding all valid suggestions to the vocabulary, and correcting any errors found, we usually run the resulting automaton again through the AutoAnalysis program, with similar parameters. Since states and suffixes change, the output of this second run usually contains many suggestions that were not present in the first run, and a good fraction of the new suggestions is valid. Typically, we can go through two or three iterations of this process before the percentage of valid suggestions in the output becomes too small to justify the cost of manual checking.

### 3.3 Lexical classes

A third function of the AutoAnalysis program is to analyze and describe the lexical structure of the wordlist.

In particular, we say that a state $t$ of the automaton corresponds to a *lexical class* if it can be reached from the initial state by a certain minimum number of distinct paths (also a program parameter), and at least one of these paths does not go through another lexical class. The set of *lexical radicals* of a state $t$ is the set of its prefixes which are spelled without going through other lexical classes. These two concepts are useful in identifying approximations to the true lexical categories of the language, and can be used as a basis for determining the parameters for the program IdealClasses, described in Section 5.

## 4 Experimental results

We have applied the AutoAnalysis program to several vocabularies, with varying degrees of thoroughness. We report here the experimental results achieved for Portuguese and Italian, which belong to the Romance group, and for English.

### 4.1 Portuguese

The Portuguese vocabulary we started from (courtesy of TTI Tecnologia Ltda. of São Paulo) had 206,784 words and was collected by processing texts published in newspapers followed by some manual verification. Its coverage of stems and non-verb derivatives was somewhat inconsistent, but its verb conjugations were mostly complete, and it contained very few erroneous words. The statistics for its automaton are:

| | |
|---|---|
| Words | 206,784 |
| States | 15,305 |
| Final states | 2,247 |
| Valid transitions | 41,824 |

The *unproductive states* analysis, with threshold $p_0 = 1$, produced a list of 196 states, each of them traversed by only one word. There were 120 distinct words in this set, shown in Figure 1 (notice that one word can contribute with several unproductive states). Most of these words fall into one of the three classes:

- words spelled incorrectly (for example *corrijindo* instead of *corrigindo*, and *intrasmissível* instead of *intransmissível*);
- words which have unusual forms (for instance words of foreign origin like *celsius* or *kibutz*);
- words which appear without corresponding inflected forms (for example, the noun *angústia* appears without its plural *angústias*, the verb form *conviéramos* appears without its many related forms, the word *intransmissíveis* appears because of the wrong form of its singular version).

We ran the same analysis a second time with $p_0 = 2$, meaning that states with up to two paths going through them were considered unproductive. This time the program reported 1244 unproductive states due to 1452 words. Most of the new suggestions were perfectly valid words which did not have other similar forms other than their plurals, like the pairs *barítono* and *barítonos* or *jurisprudência* and *jurisprudências*. Even so, we found in this run some incorrect pairs of words like *beastialidade* and *beastialidades* (instead of *bestialidade/s*), and several missing singular or plural forms, such as *cúria* and *dilúvios*.

For the analysis of *similar states*, we tried several values of the parameters for the same automaton. In a typical experiment, we required

$$\begin{aligned} |S_1 \setminus S_2| &\leq 1 \\ |S_2 \setminus S_1| &\leq 1 \\ |S_1 \cap S_2| &\geq 10 \\ |P_1| \cdot |S_2 \setminus S_1| + |P_2| \cdot |S_1 \setminus S_2| &\leq 10 \end{aligned}$$

With these values of the parameters, the program identified 1,351 pairs of similar states and suggested a list of 3,307 words for inclusion. About 25% of these suggestions were valid and should be included in the vocabulary. As expected, most of the suggested valid words were missing inflected forms of some words already present in the vocabulary, but which were not used in the material, the base for the word collection. On the other hand, many of the suggested words were non-existent "regular" forms of irregular verbs (for instance, *pressuporam* instead of *pressupuseram*). Finally, many of the suggestions were rather absurd and resulted from some unexpected state pairs being considered similar. For example, the suggestions

| | | | |
|---|---|---|---|
| áfrica | demograficamente | júpiter | povaréu |
| álcool | despreparo | jardinagem | prússia |
| abdômen | desprestígio | juniores | prefácio |
| abracadabra | dezenove | kaiser | preparativos |
| acovardamento | dezoito | kelvin | pretificássemos |
| advocacia | diametralmente | ketchup | quatorze |
| agronomia | dilúvio | kibutz | quiçá |
| alvenaria | doravante | kitsch | raízes |
| angústia | escombros | líbano | reinterpretação |
| anonimamente | esguelha | lucíferes | rusticidade |
| antropofagia | estenografia | múndi | sífilis |
| apetrechos | experdirmos | malgrado | sarapatel |
| autoconfiança | fênix | mantimentos | satanás |
| autocontrole | fealdade | metafisicamente | similitude |
| bônus | gasogênio | miosótis | sobremaneira |
| beisebol | guache | nápoles | soslaio |
| brutamontes | hércules | núpcias | status |
| buraqueira | hindustani | ninguém | têmporas |
| cócegas | hipersensibilidade | oásis | talquinho |
| catolicismo | ilicitamente | obliquamente | tecnicolor |
| celsius | ilogicamente | oeste | tibieza |
| chimarrão | incessantemente | outrossim | tiracolo |
| clímax | indianópolis | pâncreas | tranfiguração |
| conviéramos | inglaterra | pêsames | transfigurações |
| convulsivamente | intransmissíveis | píton | transitoriedade |
| copacabana | intrasmissível | parabéns | trienal |
| corrijindo | intrepidez | patavina | trigonometria |
| cristianismo | intrinsecamente | pentecostes | voleibol |
| déficit | ipanema | pmdb | xérox |
| debaixo | júnior | porventura | zênites |

Figure 1: List of words responsible for unproductive states for the Portuguese vocabulary (parameter $p_0 = 1$).

*internacionalicamente* and *automatidade* were listed because the prefixes *internacionali-* and *automati-* take the automaton to two states, both with 57 suffixes, identical except for the suffixes *-dade* and *-camente*.

It should be noticed that there are many doubtful cases when it is not clear whether a derived form exists or not, like for instance *acobertante* from the verb *acobertar*. In any case, the final decision must be taken by an expert.

## 4.2 Italian

We applied our toools to an Italian vocabulary (courtesy of David Vincenzetti from the University of Milan) which had 61,183 words and was considerably less complete and systematic than the Portuguese one, especially with regard to inflected forms and verb tenses. Under normal circumstances, such an incomplete vocabulary probably should not be even considered. However it was the only Italian dictionary in public domain we found available on the network (Italian is one of the languages some of us are fluent in). This incompleteness of the Italian vocabulary explains why the corresponding automaton is proportionately bigger than the Portuguese one:

| | |
|---|---|
| Words | 61,183 |
| States | 10,273 |
| Final states | 1,700 |
| Valid transitions | 25,941 |

In this case we went through several iterations of the debugging cycle, checking the program's suggestions and updating the automaton each time. The *unproductive state* analysis (in this case, states which are used by up to three words) produced (in successive runs) 2657 "suspicious" words. About 150 of these words were obvious misspellings (for example, *immmediata* instead of *immediata*, and *caffe* instead of *caffè*). Other 1400 words were valid, but their appearance in the list revealed the omission of related forms (for example, *armistizio* appeared in this list because it plural *armistizi* was missing). Altogether, about 58% of the suggestions uncovered real errors in the vocabulary.

The analysis of *similar states* was carried out ten times with several values of similarity parameters. About 230.000 suggestions of words to be included in the vocabulary were produced. Visual processing of these suggestions showed that at least 31% of them were valid. As in the case of Portuguese, this number does not include many suggested words whose validity we were unable to decide.

As a partial result, the vocabulary grew from the original 61,183 words to 134,090 words (119% increase!). It is interesting to notice that the growth of the automaton was much more modest:

| | |
|---|---|
| Words | 134,090 |
| States | 12,496 |
| Final states | 2,685 |
| Valid transitions | 37,611 |

### 4.3 English

We also tested our tools on an inflected English vocabulary of 104,216 words, the result of merging several public-domain word lists retrieved by FTP from sites around the world. Not surprisingly, the automaton was substantially bigger (in absolute terms, and in relation to the size of the vocabulary) than the automata for Portuguese and Italian:

| | |
|---|---|
| Words | 104,216 |
| States | 47,116 |
| Final states | 7,484 |
| Valid transitions | 99,472 |

Debugging of this vocabulary is still unfinished. The first run of the *unproductive states* analysis listed 1825 "suspicious" words, of which 409 were considered to be inappropriate during the manual check. Most of these words were proper nouns and their derivatives (like *lilliput* and *huckleberry*) or foreign terms (like *aquafortis*, *blitzkrieg*, and *mozzarella*). We did get however several misspellings, such as *arbritrary*, *apothecarcaries*, *efflrescent*, and *insugently*. Moreover, the listed words which were actually valid revealed indirectly the omission of 2269 related forms. Altogether, 62% of the words that were checked manually led to real bugs in the vocabulary.

We attempted to run the analysis of *similar states* several times, with increasingly strict similarity criteria, but the output lists were usually too big (several hundreds of thousands of words), and the runs had to be aborted. Finally, we managed to get a list with "only" 10,000 suggestions, by declaring "similar" only those pairs of states with at least 6 suffixes in common, and differing by no more than 20 words. This list is still being checked manually, and about 30% of its entries seem to be valid words that are missing from the vocabulary.

The difficulty in getting a reasonable list of suggestions was to be expected. English is not a strongly inflected language, and the whole concept of similar states depends on the existence of several related forms of a word.

## 5 The IdealClasses program

We are presently developing and testing the program IdealClasses, whose purpose is to perform a more sophisticated version of the *similar states* analysis, taking into account information on the structure of the language that is provided by the user.

The IdealClasses program will take as input an acyclic automaton, and a collection of *classes* $G_1, G_2, \ldots, G_n$ — sets of word endings which are supposed to correspond to important lexical categories of the language. In the case of a Romance language, for example, each $G_i$ could be the set of endings for a specific verb conjugation class, or the set of gender and number endings for a specific category of nouns.

The main task of the program is to determine, for each state $t$ of the automaton, a subcollection $\mathcal{G}(t) = \{G_{i_1}, G_{i_2}, \ldots\}$ of these classes, which are approximately contained in the suffix set $S(t)$ of $t$. The implication is that every prefix string leading to state $t$ looks like a "stem" of the language that belongs to all those classes, and can be completed with any of the corresponding endings.

For instance, in the case of an English vocabulary, the program might assign these three classes to the state $t$ that is reached by the prefix *ration*-:

1. the regular verb endings: $G_1 = \{$-, -*s*, -*ed*, -*ing*$\}$;
2. the regular noun endings: $G_2 = \{$-, -*s*, -*'s*, -*s'*, -*like*$\}$;
3. the *noun+al* adjective endings: $G_3 = \{$-, -*al*, -*ally*$\}$.

By including these classes in $\mathcal{G}(t)$ the program is "explaining" many of the words that use state $t$, like *rationed* and *rationally*. Note that these classes do not account for the word *rationalized*, even though -*alized* is a suffix of $t$. However, assuming that the program was given also the alternative verb class

$$G_4 = \{\text{-}e, \text{-}es, \text{-}ed, \text{-}ing\}$$

then this class will get included in $\mathcal{G}(u)$, where $u$ is the state reached through the prefix *rationaliz*-. This assignment will then "explain" the word *rationalized*.

Thus, every assignment of classes $\mathcal{G}(t)$ to the states of the automaton defines an *idealized* version of the vocabulary, consisting of the words

$$\bigcup_{t \in Q} P(t) \cdot \left( \bigcup_{G \in \mathcal{G}(t)} G \right)$$

where $Q$ is the set of all states, and $P(t)$, as before, is the set of prefixes of state $t$. Moreover, the assignment also defines a factoring of each word in this idealized language into a "stem" (a prefix of some state) and an "ending" (a string from some $G_i$); and therefore also assigns the word to a specific class. Note that the classes $G_i$ need not be disjoint, and that certain words (like *rations* in English, or *descendo* in Portuguese) may be factored and classified in more than one way.

## 5.1 Assigning classes to states

If languages were perfectly regular, and the input vocabularies were complete and error-free, the IdealClasses algorithm would be very simple: merely assign to each state $t$ the set $\mathcal{G}(t)$ of all classes $G_i$ that are contained in the suffix set $S(t)$.

Unfortunately, this algorithm is not suitable for real-world data, since the omission of a single word from the vocabulary could prevent a whole class $G_i$ from being included in $\mathcal{G}(t)$. Therefore, the program will include a class $G_i$ in $\mathcal{G}(t)$ even if it is not contained in $S(t)$, as long as the difference $G_i \setminus S(t)$ is smaller then a user-specified threshold.

As part of the class-matching process, IdealClasses also builds a finite acyclic automaton $\mathcal{N}$ that accepts the idealized version of the input vocabulary. Initially, $\mathcal{N}$ is empty. The program examines states of the input automaton in topological order, from the final state towards the root. For each state $t$, the program first determines the matching classes $\mathcal{G}(t)$; then it adds to $\mathcal{N}$ a new state $\iota(t)$ that recognizes the "idealized" version $\mathcal{I}(t)$ of $S(t)$; that is,

$$\mathcal{I}(t) = \mathcal{R}(t) \cup \bigcup_{G \in \mathcal{G}(t)} G$$

where

$$\mathcal{R}(t) = \bigcup_{a \in \Sigma} \{a\} \cdot \mathcal{I}(t^a)$$

and $t^a$ stands for the state reached from $t$ through the letter $a$.

We are currently experimenting with more complex criteria for choosing the matching classes $\mathcal{G}(t)$. For instance, in the case of highly inflected languages it seems a good idea to consider the classes $G_i$ in order of decreasing size, and only add a class $G_i$ to $\mathcal{G}(t)$ if $G_i \setminus \mathcal{I}(t)$ is bigger than some threshold.

## 5.2 Using the IdealClasses tool

Many applications of wordlists require that the words be marked with lexical category (verb, noun, etc.), and sometimes with even finer categories (transitive, intransitive, reflexive, etc.). We hope that the IdealClasses program will be a useful aid for the task of adding such information to a plain wordlist.

The input classes may be obtained from grammar books or other linguistic sources, or may be extracted from the wordlist itself, with the help of AutoAnalysis, without any knowledge of the language. Indeed, it is conceivable that IdealClasses may turn out to be a useful tool for linguistic research.

In any case, we expect IdealClasses to be useful as a debugging tool. By subtracting the idealized vocabulary from the original one, we will get a list of words whose sets of endings do not fit any of the given classes. Misspellings and foreign words are thus likely to appear in this list. The preliminary experiments in this direction are quite encouraging.

The program should be useful also as a tool for splitting a word list into sub-lists according to lexical criteria (say, verbs and non-verbs), each of them more regular than the original list, and hence easier to debug.

# 6 Future work and conclusions

Our experience with the AutoAnalysis tool suggests that it would be even more effective if it were combined with MaintainAutomaton and made interactive. This improved tool would repeatedly look for a possible problem in the automaton (an unproductive state, or a pair of similar ones), present the corresponding suggestions to the user for manual verification, and immediately incorporate any corrections in the automaton. The new tool would also allow the user to interactively change its parameters, such as the definition of similar states.

It also seems desirable to provide the *similar states* algorithm with some means to automatically detect and ignore some obviously bad suggestions, such as words that were previously checked and rejected by the user (which we currently remove with the Unix tool `comm`, before the manual check), and words with invalid letter combinations, such as three consecutive occurrences of the same consonant. We believe that this goal can be implemented very efficiently by maintaining two additional automata: one (very compact) representing the universe $U$ of all letter sequences of length $\leq l_{max}$ with no invalid trigrams, and one encoding the set $R$ of all previously rejected suggestions. Then, for example, we could redefine two states to be similar only if they can be made equal by the addition of up to $k$ words from $U \setminus R$. This change alone could easily double the tool's efficiency (ratio of valid suggestions to total suggestions).

As a final remark, we can conclude that finite automata provide an effective model in debugging large natural language vocabularies, even when such simple tools as our AutoAnalysis program are used.

# References

[1] Appel, A. W., Jacobson, G. J., "The wold's fastest Scrabble program." *Commun. ACM*, **31**(5), 572-578, 1988.

[2] Gross, M., Perrin, D. (eds.), *Electronic Dictionaries and Automata in Computational Linguistics*, Lecture Notes in Computer Science 377, Springer-Verlag, 1989.

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

[4] Kowaltowski, T., Lucchesi, C. L., Stolfi, J., "Minimization of binary automata", in this issue.

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

[6] McIlroy, M. D., "Development of a spelling list", *IEEE Trans. Comm.*, **30**(1), 91-99, 1982.

[7] Nelson, G. (ed.), *System Programming with Modula-3*, Prentice Hall, 1991.