INTERNATIONAL JOURNAL OF SPEECH TECHNOLOGY 6, 259–275, 2003

# Efficient Development of Lexical Language Resources and their Representation

MATEJ ROJC AND ZDRAVKO KAČIČ
*University of Maribor, Faculty of Electrical Engineering and Computer Science, Maribor, Slovenia*
matej.rojc@uni-mb.si
kacic@uni-mb.si

**Abstract.** Statistical approaches in speech technology, whether used for statistical language models, trees, hidden Markov models or neural networks, represent the driving forces for the creation of language resources (LR), e.g., text corpora, pronunciation and morphology lexicons, and speech databases. This paper presents a system architecture for the rapid construction of morphologic and phonetic lexicons, two of the most important written language resources for the development of ASR (automatic speech recognition) and TTS (text-to-speech) systems. The presented architecture is modular and is particularly suitable for the development of written language resources for inflectional languages. In this paper an implementation is presented for the Slovenian language. The integrated graphic user interface focuses on the morphological and phonetic aspects of language and allows experts to produce good performances during analysis. In multilingual TTS systems, many extensive external written language resources are used, especially in the text processing part. It is very important, therefore, that representation of these resources is time and space efficient. It is also very important that language resources for new languages can be easily incorporated into the system, without modifying the common algorithms developed for multiple languages. In this regard the use of large external language resources (e.g., morphology and phonetic lexicons) represent an important problem because of the required space and slow look-up time. This paper presents a method and its results for compiling large lexicons, using examples for compiling German phonetic and morphology lexicons (CISLEX), and Slovenian phonetic (SIflex) and morphology (SImlex) lexicons, into corresponding finite-state transducers (FSTs). The German lexicons consisted of about 300,000 words, SIflex consisted of about 60,000 and SImlex of about 600,000 words (where 40,000 words were used for representation using finite-state transducers). Representation of large lexicons using finite-state transducers is mainly motivated by considerations of space and time efficiency. A great reduction in size and optimal access time was achieved for all lexicons. The starting size for the German phonetic lexicon was 12.53 MB and 18.49 MB for the morphology lexicon. The starting size for the Slovenian phonetic lexicon was 1.8 MB and 1.4 MB for the morphology lexicon. The final size of the corresponding FSTs was 2.78 MB for the German phonetic lexicon, 6.33 MB for the German morphology lexicon, 253 KB for SIflex and 662 KB for the SImlex lexicon. The achieved look-up time is optimal, since it only depends on the length of the input word and not on the size of the lexicon. Integration of lexicons for new languages into the multilingual TTS system is easy when using such representations and does not require any changes in the algorithms used for such lexicons.

**Keywords:** morphology, grapheme-to-phoneme conversion, text processing, lexicons, multilinguality, finite-state machines

## 1. Introduction

During the last decade important efforts can be identified in the development of lexical knowledge databases, including different linguistic knowledge types such as syntactic, morphological, phonological, semantic, and others. Every natural language processing system (NLP) needs to manage linguistic knowledge and a high volume of lexical data, as well as incorporate efficient computational techniques for performing linguistic analysis. Practical NLP applications require large lexical resources, but their construction is very time-consuming. The appropriate tools for rapid resource development are needed because the development of natural language processing systems must be quick and efficient. This paper describes a morphological analysis and a grapheme-to-phoneme conversion tool with a graphical interface that allows the rapid and reliable construction of lexicons. Furthermore, a procedure is described for the time and space optimal representation of both lexicons.

A language resources (LR) development tool allows experts to include, revise and validate lexical knowledge, while achieving good performance of analysis time. The linguistic knowledge included in the graphical user interface focuses on the morphological and phonetic aspects of inflectional languages, in this case the Slovenian language. The emphasis in developing this tool was to make this process as productive as possible.

Today, many text corpora resources are available in electronic form (e.g., Internet, CD-ROMs). In this work most of the needed text corpora for the construction of morphological and phonetic lexicon was available on CD-ROMs (newspaper articles) or was downloaded from the Internet. Also, texts from literature were available. The obtained text corpus consisted of about 31 million words. The tokenization of text into word tokens was performed after conversion into the text format, to obtain the list of root items for the construction of morphological and phonetic lexicons.

In the following a definition of needed text resources for the lexicons' construction will be given and the data preparation step described in more detail. The architecture of the tool will then be presented, including all the modules. The basic capabilities of the system will be discussed and the notation used will be described. A tool performance evaluation will also be reported and, finally, a conclusion will be drawn.

## 2. System Architecture for Building Morphological and Phonetic Lexicons

Figure 1 shows the system for the lexicons' development. This figure shows that the system architecture consists basically of two levels. First, the data preparation step is performed, followed by the lexicon's construction. The lexicon's construction consists of two

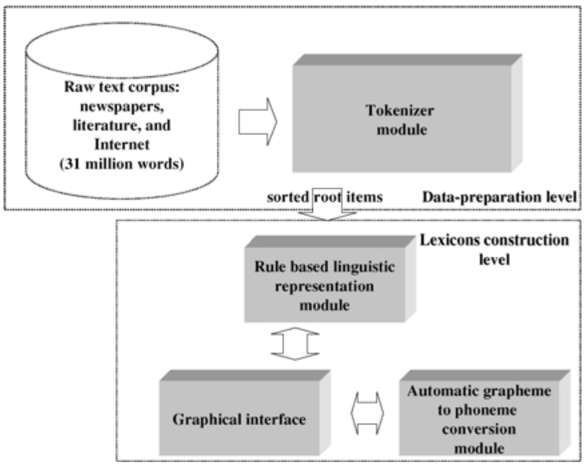

Raw text corpus: newspapers, literature, and Internet (31 million words)
Tokenizer module
sorted root items
Data-preparation level
Lexicons construction level
Rule based linguistic representation module
Graphical interface
Automatic grapheme to phoneme conversion module

*Figure 1.* System architecture for building morphological and phonetic lexicons.

modules: a rule-based linguistic module and an automatic grapheme-to-phoneme conversion module. A graphical user interface links all the modules together. A more detailed description of the system modules is given later. The architecture of the system is modular and multilingually oriented. Appropriate system modules must be adopted for using the presented platform in other languages. The tokenization module is capable of multilingual text processing (Rojc et al., 1999), and the statistical part of the grapheme-to-phoneme conversion module is also multilingual, since it uses a data-driven approach based on neural networks. The graphical user interface currently supports the Slovenian language, but can also be adapted for other languages. Almost all the modules are written in the C++ programming language. The exception is the graphical user interface, which is written in Java language using Visual J++. The whole system runs on all Windows platforms—Windows 95/98 and Windows NT/2000/XP.

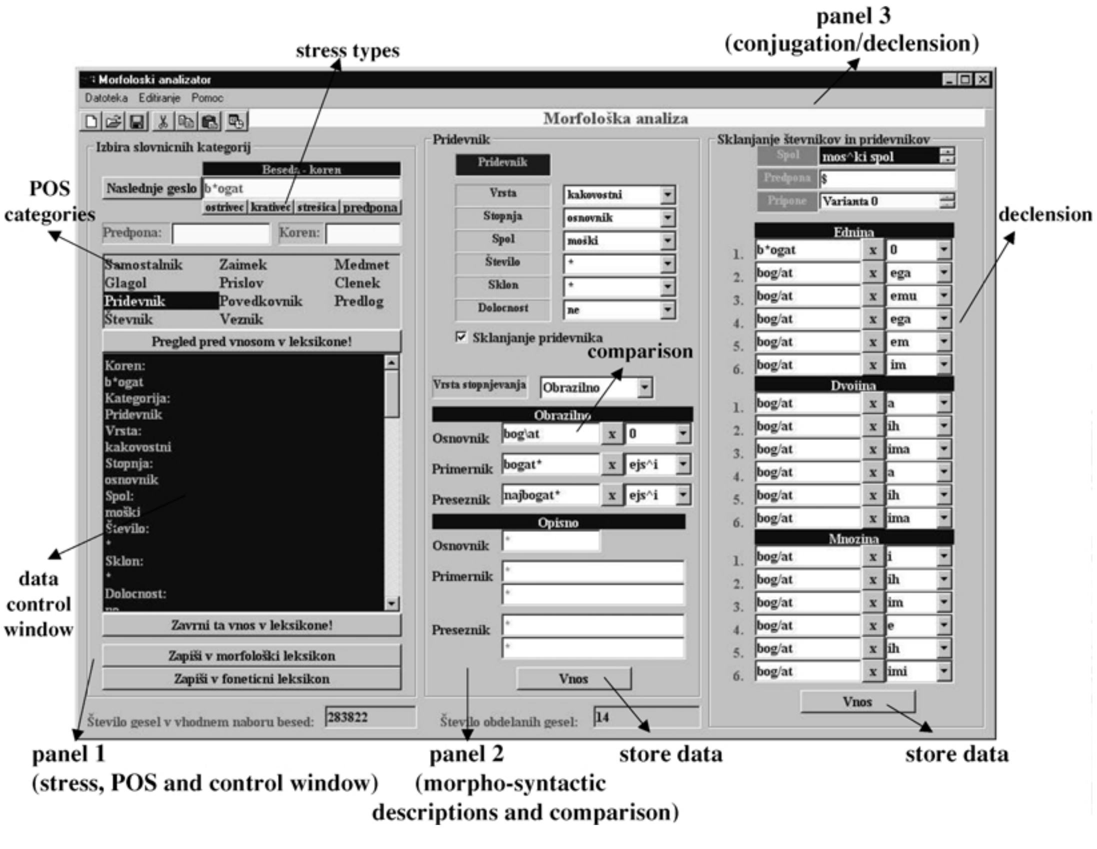
## 3. Tokenization and Word Selection Process

Some text pre-processing on the obtained text corpus must be done before using this tool for building morphological and phonetic lexicons (e.g., for the Slovenian language). Pre-processing algorithms must be highly flexible and robust because of the general, unrestricted nature of the text. As shown in Fig. 1, the input text corpus (raw ASCII text) is fed into the tokenizer module (finite-state machine) (Rojc et al., 1999), which emits hypotheses about tokens and segments the input text into words.

The tokenizer module is organized on a multilevel basis. At the lowest level the lexical scanner separates the input text into tokens. Some tokens may not be in a canonical form appropriate for constructing morphological and phonetic lexicons. In this case, the text normalization processing level breaks such tokens into their constituent words. All tokens such as date, hour, cardinal numbers and ordinal numbers are expanded into corresponding word forms during the tokenization process in the tokenizer module ('expanding text processing level'). The obtained words are sorted, and a word frequency is assigned to each one. A final list of items was defined by this procedure, using the 30,000 most frequent words in the input corpus (root forms).

## 4. Rule-Based Linguistic Representation Module

Figure 2 shows the graphical user interface of the system (lexicons construction level). This interface visually presents all the information generated by the rule-based linguistic module (e.g., adjectives). An expert also is able to perform editing, correction, and verification actions when using it. The morphological analysis represents the main part of the system. The graphical user interface consists of three panels. The first panel is fixed and is intended as a starting point for any analysis the expert needs to perform. The linguistic expert has to load existing phonetic and morphological lexicons, or create a new one. A list of all items to which phonetic and morphology information will be assigned is also loaded as input. All items on the list must be in the root form and alphabetically ordered. Then the expert manually chooses an item from the list or just types a new one. Next, the expert verifies the type and position of the stress and marks a suffix in the item if it exists. The syntactic category for the corresponding item (part-of-speech) is further chosen: noun, verb, adjective, number, pronoun, adverb, conjunction, interjection, article, and predicative. This action opens the appropriate second and third panels for corresponding parts-of-speech. Based on this selection, the rule-based linguistic representation component generates all attributes and values and enters them into their respective fields. The



Beseda - koren: b*ogat

Predpona: | Koren:

Koren:
b*ogat
Kategorija:
Pridevnik
Vrsta:
kakovostni
Stopnja:
osnovnik
Spol:
moški
Število:
*
Sklon:
*
Dolocnost:

Število gesel v vhodnem naboru besed: 283822

Pridevnik

| | |
|---|---|
| Vrsta | kakovostni |
| Stopnja | osnovnik |
| Spol | moški |
| Število | * |
| Sklon | * |
| Dolocnost | ne |

☑ Sklanjanje pridevnika

Vrsta stopnjevanja: Obrazilno

| Obrazilno | | | |
|---|---|---|---|
| Osnovnik | bog\at | x | 0 |
| Primernik | bogat* | x | ejs^i |
| Preseznik | najbogat* | x | ejs^i |

| Opisno | |
|---|---|
| Osnovnik | * |
| Primernik | * |
| | * |
| Preseznik | * |
| | * |

Število obdelanih gesel: 14

Sklanjanje števnikov in pridevnikov

Spol: mos^ki spol
Predpona: $
Pripone: Varianta 0

| | | | |
|---|---|---|---|
| | Ednina | | |
| 1. | b*ogat | x | 0 |
| 2. | bog/at | x | ega |
| 3. | bog/at | x | emu |
| 4. | bog/at | x | ega |
| 5. | bog/at | x | em |
| 6. | bog/at | x | im |
| | Dvoiina | | |
| 1. | bog/at | x | a |
| 2. | bog/at | x | ih |
| 3. | bog/at | x | ima |
| 4. | bog/at | x | a |
| 5. | bog/at | x | ih |
| 6. | bog/at | x | ima |
| | Mnozina | | |
| 1. | bog/at | x | i |
| 2. | bog/at | x | ih |
| 3. | bog/at | x | im |
| 4. | bog/at | x | e |
| 5. | bog/at | x | ih |
| 6. | bog/at | x | imi |

*Figure 2*. Graphical user interface of the system for building morphological and phonetic lexicons.

linguistic expert then verifies all the values and corrects them if needed. The data control window serves for verification of the generated data. The expert can define the comparative forms of adjectives and adverbs and the data for lemmas using the second panel. The third panel is used for the declension or conjugative forms of lemmas.

Currently this component works for the Slovenian language only, but could be adapted for other languages by integrating the appropriate linguistic rules. Slovenian, like other Slavic languages, is an inflectional language, and the linguistic representation of a word depends on complex contextual factors. Most general linguistic rules were integrated into this component. Because there are many exceptions in the Slovenian language, a linguistic expert's verification is required following the automatic generation of all forms for the current item's linguistic category (POS—part of speech). All items for processing are in their root form and, as a result, all possible forms of the corresponding item assigned with detailed linguistic information are automatically defined. All the obtained results are added to the existing morphological lexicon in the prescribed format after the linguistic expert verifies them.

Some basic actions that must be performed for specific part-of-speech categorization are defined below:

- adjective: the comparison is performed automatically and a conjugation/declension panel must be activated,
- nouns: appropriate declension must be chosen and a conjugation/declension panel activated,
- verbs: several panels for building various verb forms must be activated and verified after automatic generation (infinitive, supine, participle, verb conjugation, etc.),
- number: the expert determines gender, number, etc. for the root form and activates a conjugation/declension panel,
- pronoun: its type, gender, person, number and case are determined for the root form, and a conjugation/declension panel also is activated,
- adverb: comparison is performed automatically and the type chosen,
- conjunction: the type is determined,
- interjection: the type is determined,
- article: the type is determined, and
- predicative: the type and case are determined.

When the expert verifies the content of the second panel, the information is saved in memory by pushing the 'store data' button. The expert then moves to the third panel (conjugation/declension), where conjugation/declension forms of the root item are automatically generated (adjective, nouns, verbs, and numbers). In the Slovenian language there are many exceptions that cannot always be correctly interpreted by the rule-based linguistic representation component. Sometimes the stress changes the position and type in the word during the conjugation/declension process. Manual corrections by an expert are needed because this is very hard to predict correctly using rules.

## 5. Automatic Grapheme-to-Phoneme Conversion Module

The words in their grapheme representation have to be mapped onto their phonetic representation, i.e., into their pronunciation for speech recognition and text-to-speech synthesis. The availability of pronunciation lexicons in their canonical form (a single phonemic representation for each word) is very important in order to be able to build grapheme-to-phoneme models. In some languages this form can be derived from the grapheme form by a set of pronunciation rules, but for many languages (e.g., for the Slovenian language) this relationship is complex and is usually handled by manually produced pronunciation lexicons.

An automatic grapheme-to-phoneme conversion module was added to make it easier for a linguistic expert to develop the pronunciation lexicon. It consists of two parts. The first part uses a rule-based approach, and the second is based on a data-driven approach, using neural networks. The first part is intended for use at the initial stage of phonetic lexicon construction, when there is no material for training a neural network. First, a rule-based stress assignment is done (in cases where grapheme-to-phoneme conversion is dependent on the stress position and stress type—e.g., as in the Slovenian language), followed by a grapheme-to-phoneme conversion procedure. Rule-based syllabification on the phonetic transcriptions obtained can then be performed. The result is verified and, if needed, corrected by the linguistic expert. The generated phonetic transcriptions are added to the phonetic lexicon after they are corrected.

The data-driven approach using a neural network can be used as soon as we have enough data in the phonetic lexicon generated by the process described above. The neural network, which was taken as the basis for this part, is based on a method used and described in the

SNNS (Stuttgarter Neuronaler Netz Simulator SNNS) (Zell, 1994), which provides different training methods for a variety of applications. The data preparation, generation of training patterns and the training of a neural network are done completely automatically. A standardized alignment between the letters in an entry of the lexicon and the phones in corresponding pronunciation form must be performed before building models.

### *5.1. String Alignment*

In many languages the number of phones in a word's pronunciation and the number of letters in an orthographic transcription of a word are not a one-to-one match. Letters usually can be mapped either to zero, one, or two phones. When letters in some contexts correspond to no phone, they are marked with an empty symbol (_epsilon_). The alignment task actually becomes an introduction of epsilons into the phonemic representation so that it matches the length of the grapheme representation. The explicit listing of the phones (or multi-phones) to which each letter in the alphabet may correspond, irrespective of the context, is defined before doing string alignment. This task can be done in an interactive process over the training set from the phonetic lexicon. In the list of allowable mappings, the vowels have a much longer list of potential phones. The input of the algorithm used is the list of allowable mappings and input lexicon. The probability of one grapheme G matching with phoneme P is estimated during processing, and DTW is used to introduce epsilons at positions maximizing the probability of the word's alignment path. Once the dictionary is aligned, the association probabilities can be computed again, and so on until convergence.

Algorithm for alignment of strings:

```
//initialize prob(G,P) the probability of G matching P
1. for each word$_i$ in training_set
   counting all possible G/P associations for all
   possible epsilon positions in the phonetic
   transcription using DTW.

//EM loop
2. for each word$_i$ in training_set
   alignment_path = arg max $\prod_{i,j}$ prob($G_i$,$P_j$)
   compute new_p(G,P) on alignment_path

3. if(prob != new_p) goto 2
```

The grapheme-to-phoneme conversion is performed in two steps. In the first, stress marks are inserted. In the second, the graphemes are converted into phonemes, and the syllable breaks are inserted in the phoneme string. The neural networks are constantly learned off-line during the lexicons' development and then integrated into the grapheme-to-phoneme conversion module to increase their performance. A multilayer perceptron (MLP) feedforward network with one hidden layer was used as the basic neural network model. A back propagation algorithm was chosen as the learning algorithm for both networks.

### *5.2. Syllabification*

Marking syllables is a very important and necessary operation in text-to-speech synthesis systems because in many languages the pronunciation of phonemes is a function of their location in the syllable, relative to the syllable boundaries. Phoneme location in the syllable also has an important role for the duration of the phone and represents significant information for any module that predicts segmental duration in a TTS system.

The word "multilingual" describes a system which uses common algorithms for multiple languages. A collection of language-specific TTS systems cannot be considered as a multilingual system. In a multilingual TTS system all the language- specific information should be stored in data tables, and all the algorithms should be shared among all languages. In accordance with this goal, the following subsections present the construction of a syllabification model using weighted finite-state transducers (Kiraz and Möbius, 1998). It can be used for any language, but this paper illustrates its use for the Slovenian language. The syllabification process is implemented as a weighted finite-state transducer, which was constructed using marked syllables in the *SIflex* dictionary (60,000 entries) (Rojc and Kačič, 2000). The weights of the corresponding weighted finite-state transducer were determined according to the statistical results of processing syllable structures and frequencies. The frequencies of the onset, nucleus, and coda types were converted into weights that are associated with the corresponding transitions of the finite-state transducer.

***5.2.1. Syllable Structure of the Slovenian Language.*** The Slovenian language allows complex consonant clusters in the onset and coda of syllables. The classes of Slovenian phones are presented in Table 1. Some

*Table 1.* Classes of Slovenian phones.

| Class | Description | Slovenian phones |
|---|---|---|
| P | unvoiced stops | p t k |
| B | voiced stops | b d g |
| S | unvoiced fricatives | f s S x |
| Z | voiced fricatives | z Z |
| N | nasals | m n |
| L | liquids, glides | l r v j |
| V | vowels | i: e: E: a: O: o: u: i E a O u @ |

*Table 2.* Slovenian language allows up to 3 consonants in the onset of the syllable and 4 consonants in the coda of the syllable.

| Onsets | | Codas | |
|---|---|---|---|
| Class | Clusters | Class | Clusters |
| PLL | klj | LSP | jst |
| SPL | skr | LSPN | jstn |
| SNL | Snj | SPP | stk |
| LLL | vrv | BLL | brv |
| ZBL | zbr | BLP | brt |

examples of complex consonant clusters are given in Table 2. Statistical results show that the Slovenian language allows up to four consonants in the onset and up to four consonants in the coda of the syllable. Determining the syllable boundary correctly is important because the pronunciation of most phonemes is a function of their position in the syllable. Syllable boundaries also influence phoneme duration. The syllabification process is implemented as a weighted finite-state transducer (WFST) that is constructed from a list of syllables. The weights of the finite-state transducer were determined using frequencies of onset, nucleus, and coda types obtained from the statistical processing of training data (syllables). They play a significant role in obtaining the correct syllabification, especially in the case of consonant clusters.

***5.2.2. Finite-State Automata and Finite-State Transducers.*** Finite-state machines are used in many areas of natural language processing. Their use is motivated mainly by considerations of space and time efficiency from the computational point of view. Linguistically, the use of finite-state machines (Mohri, 1996; Berstel and Reutenauer, 1988) is very convenient because they allow an easy description for most of the relevant local phenomena in a language. They also provide compact representation of the language-specific lexical rules needed for knowledge representation. These features of finite-state machines are of major importance in the field of multilingual text processing, large scale lexical representation, etc. This paper presents a multilingual syllabification module using weighted finite-state transducers and later a string-string finite-state transducer for representation of large scale morphological and phonetic lexicons. An approach is presented for compiling such lexicons into finite-state transducers that are their time and space optimal representation. The effects of using finite-state transducers for representation of external natural language resources are a great reduction in the memory required by the lexicons and the optimal access time (required for obtaining information) that is independent of the size of the lexicons.

*Finite-State Automata (FSA).* Finite-state automata (FSA) (Mohri, 2000) can be seen simply as an oriented graph with labels on each arc. Fundamental theoretical properties make FSAs very flexible, powerful and efficient. FSAs can be seen as defining a class of graphs and also as defining languages (Aho et al., 1974 ; Hopcroft and Ullman, 1979; Kuich and Salomaa, 1986).

*Definition.* A finite-state automaton A is a 5-tuple $(\Sigma, Q, i, F, E)$ where $\Sigma$ is a finite set called the alphabet, Q is a finite set of states, $i \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and $E \subseteq Q \times (\Sigma \cup \{\in\}) \times Q$ is the set of edges. FSAs have been shown to be closed under union, Kleen star, concatenation, intersection and complementation, thus allowing for natural and flexible descriptions. In addition to their flexibility due to their closure properties, FSAs can also be turned into canonical forms that allow for optimal time and space efficiency.

*Finite-State Transducer (FST).* FSTs (Mohri, 2000) can be interpreted as defining a class of graphs, a class of relations on strings, or a class of transductions on strings. Based on the first interpretation, an FST can be seen as an FSA in which each arc is labeled by a pair of symbols rather than by a single symbol (Crochemore, 1986).

*Definition.* A finite-state transducer T is a 6-tuple $(\Sigma_1, \Sigma_{2,} Q, i, F, E)$ such that:

- $\Sigma_1$ is a finite alphabet, namely the input alphabet,
- $\Sigma_2$ is a finite alphabet, namely the output alphabet,
- Q is a finite set of states,
- $i \in Q$ is the initial state,
- $F \subseteq Q$ is the set of final states,
- $E \subseteq Q \times \Sigma_1^* \times \Sigma_2^* \times Q$ is the set of edges.

As with FSAs, FSTs are also powerful because of their various closure and algorithmic properties. In this paper the following conventions are followed when describing an FST: final states are depicted by a bold circle; $\varepsilon$ represents an empty string; and the initial state (labelled 0) is the leftmost state appearing in the figure.

***5.2.3. Weighted Finite-State Model for Multilingual Syllabification.*** Let's mark syllabification finite-state transducer as $T_{\text{syll}}$. Such a transducer contains one additional symbol, '-' in the alphabet, that marks the syllable boundaries. The correct syllable boundary is very important for TTS systems. The duration prediction of segments depends on whether the segment is in the coda, nucleus, or onset of a given syllable. Another feature that can be used is that the finite-state transducers (FST) are bi-directional. If the transducers are inverted, the system that maps a phonetic string back to its orthographic string is obtained. Such a transducer is actually a de-syllabifier $T_{\text{syll}}^{-1}$. The syllabifier presented in this paper is implemented as a weighted finite-state transducer. The corresponding weights were determined during training using the *SIflex* phonetic lexicon for the Slovenian language.

***5.2.4. Construction of Onset, Nuclei and Coda Automata.*** The construction of a syllabification transducer for the Slovenian language is presented in the following. The SIflex lexicon was used as the training material. The lexicon already contains some syllabified phonological words, for instance:

/afrika
/a: - f r i - k a
af*er
a - f *E r
ag\ent
a - g \E n t
. . .

In the phonetic lexicon the phonetic transcription is written using SAMPA symbols for the Slovenian language (Sampa, 1998). In the above entries the symbol '-' marks syllable breaks, and the symbols '*', '/', and '\' are stress marks, corresponding to the three different types of stress that exist in the Slovenian language (long and narrow, long and wide, short and wide).

First, a list of all the syllables in the training database was obtained. Four syllables were obtained for the Slovenian word [pacientov (*patient's*)] with phonetic transcription [$p\,a - ts\,i - j\,{*}E\,n - t\,o\,w$]. Each syllable in the list of syllables is then split into plausible onsets, nuclei, and codas. As an example, the syllable components obtained for the phonological word [$p\,a - ts\,i - j\,{*}E\,n - t\,o\,w$] are presented in Table 4.

Then the sets of plausible onsets, nuclei, and codas are computed with their occurrence frequencies. Statistics for each onset, nucleus and coda in the training data are actually performed regardless of the context. The set of Slovenian nuclei found in the *SIflex*

*Table 3.* The sets of Slovenian nuclei found in the SIflex dictionary, together with the number of observations (f). Weights for the transitions between states of the automaton for each nucleus are obtained by calculating the reciprocal of each nucleus type's frequency.

| Nucleus | f | $1/\text{f} \cdot 10^3$ |
|---|---|---|
| a | 7227 | 0.13837 |
| i | 5940 | 0.16835 |
| O | 6017 | 0.166196 |
| O: | 1881 | 0.531632 |
| io: | 3 | 333.33 |
| E | 6314 | 0.158378 |
| E: | 2156 | 0.463822 |
| u | 2002 | 0.499501 |
| i: | 5819 | 0.171851 |
| o: | 4433 | 0.225581 |
| @ | 2915 | 0.343053 |
| u: | 2277 | 0.439174 |
| a: | 7799 | 0.128222 |
| e: | 5665 | 0.176523 |
| @r | 836 | 1.19617 |

*Table 4.* Syllable structure for the Slovenian word "pacientov" (*patient's*).

| Onset | Nucleus | Coda |
|---|---|---|
| p | a | |
| ts | i | |
| j | *E | n |
| t | o | w |

dictionary is given in Table 3. Each of the three sets (onsets, nuclei and codas) is compiled into a weighted finite-state transducer by taking the disjunction of its members. The corresponding frequencies are converted into weights by taking their reciprocals. The result of this module is three finite-state automata (acceptors), marked as: $A_o$ (onsets), $A_n$ (nuclei), and $A_c$ (codas):

$$A_o = \bigcup_{(o,f)\in\xi} o\langle 1/f\rangle \tag{1}$$

$$A_n = \bigcup_{(o,f)\in\psi} n\langle 1/f\rangle \tag{2}$$

$$A_c = \bigcup_{(o,f)\in\zeta} c\langle 1/f\rangle \tag{3}$$

In these formulae the values in the angle brackets represent the corresponding weights associated with the preceding symbol in an expression where $\xi$ represents the set of onsets, $\psi$ the set of nuclei and $\zeta$ the set of codas.

***5.2.5. The Phonotactic Automaton.*** The following presents the language-dependent step. The syllabic phonotactics can be described for the Slovenian language by the extended regular expression:

$$\begin{aligned} A_{\text{phonem}} &= \text{Opt}(A_o)A_n\,\text{Opt}(A_c) \\ \text{Opt}(A_o) &= A_o \cup \varepsilon \\ \text{Opt}(A_c) &= A_c \cup \varepsilon \end{aligned} \tag{4}$$

Here, Opt represents the optional operator, and $\varepsilon$ denotes the empty string. The above equation accepts an optional onset from automaton $A_o$, followed by an obligatory nucleus from automaton $A_n$, followed by an optional coda from $A_c$. This is the only language-dependent expression. Other languages may use different expressions. In some languages, such as Arabic and Syriac, the onset is an obligatory part of the syllable and not optional (Kiraz, 1998). In this case the optional operator defined in the equation must be omitted.

The syllabification automaton, denoted by $A_{\text{syll}}$, must accept a sequence of syllables, each followed by a syllable boundary marker '-' (except the last one). This can be achieved by the expression:

$$A_{\text{syll}} = A_{\text{phonem}}(-A_{\text{phonem}})^* \tag{5}$$

This automaton defines a syllable from $A_{\text{phonem}}$, that is followed by zero or more occurrences of the boundary marker '-' and a syllable from $A_{\text{phonem}}$.

***5.2.6. The Syllabification Transducer.*** The above automaton $A_{\text{syll}}$ accepts a sequence of one or more syllables, each (except the last one) followed by the syllable boundary marker '-'. What is actually needed is a finite-state transducer that will insert the syllable boundary marker '-' after each syllable except the last one. This is simply achieved by computing the identity transducer for automaton $A_{\text{phonem}}$ and replacing the above marker '-' with a mapping '$\varepsilon$:-'. The obtained syllabification transducer is then:

$$T_{\text{syll}} = \text{Id}(A_{\text{phonem}})((\varepsilon :\text{-})\text{Id}(A_{\text{phonem}}))^* \tag{6}$$

The Id operator produces the identity transducer of its argument. The transducer $T_{\text{syll}}$ is shown in Fig. 3.

***5.2.7. Post-Processing.*** As shown in Table 3, some nucleus types, like (io:) for example, have an extremely low number of observations in the training data. In this case this indicates probable erroneous entries in the training database. Corresponding weights derived from the frequencies of nucleus types must be manually corrected (Kiraz and Möbius, 1998). Hand-tuning is also important when dealing with training databases for which data are scarce. In such cases it is not feasible to use databases for training. It is possible, however, to construct a syllabification transducer manually. One of the important features is that weighted transducers are very efficient regarding mathematics. The consequence is that it is easy to perform hand-tuning. First, a partial duplicate set of $\psi$ can be created, that contains only the nuclei to be finely tuned, and in which each nucleus is paired with the adjustment in weight. Let's mark this

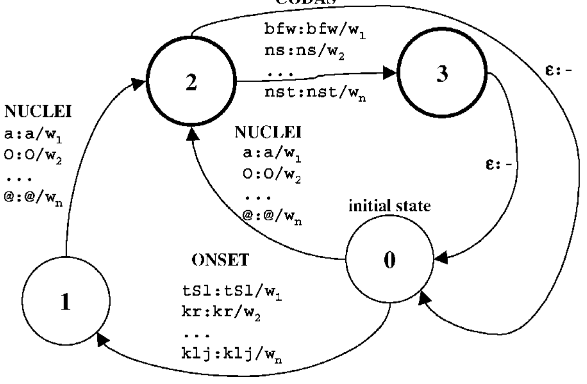


*Figure 3.* The final syllabification transducer constructed according to Slovenian syllabic phonotactics. The constructed transducer inserts a syllable boundary marker '-' after each non-final syllable.

new set as $\psi'$. The automaton for hand-tuned nuclei $A'_n$ can then be defined as:

$$A'_n = \left( \bigcup_{(n',f)\in\psi'} n'\langle 1/f\rangle \right) \bigcup \left( \bigcup_{(n,f)\in\psi-\psi'} n\langle 0\rangle \right) \quad (7)$$

As shown in Eq. (7), the first part is equal to the expression for nuclei automaton $A_n$, but computes the weights for the hand-tuned nuclei instead. To complete this set, this part is joined with the disjunction of the remaining non-hand tuned nuclei ($\psi - \psi'$), where each element in $\psi - \psi'$ is given a weight of zero.

The new automaton has now to be merged with the old one. The new automaton incorporates both $A_n$ and $A'_n$. This process can be accomplished simply by using an intersection operation of the two automata, thus obtaining the new automaton $A_{\text{nuclei}}$:

$$A_{\text{nuclei}} = A_n \cap A'_n \quad (8)$$

This explanation comes from the theory of weighted automata. When no weights are used, the intersection operation of two automata produces a third automaton, which accepts strings that both of the original two machines accept, and the weights of the common paths in the original two machines are added in the final result. This calculation is shown in Fig. 4. The computation of expressions for $A_{\text{onsets}}$ and $A_{\text{codas}}$ can be done in the same way. A new equation for the phonotactic automatons can be defined using new automata:

$$A_{\text{phonem}} = \text{Opt}(A_{\text{onsets}}) A_{\text{nuclei}}\, \text{Opt}(A_{\text{codas}}) \quad (9)$$

It is very important that hand tuning be done before the construction of $T_{\text{syll}}$. The equation $A_{\text{nuclei}} = A_n \cap A'_n$

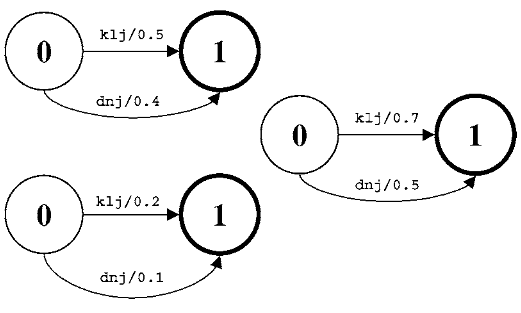


*Figure 4.* Mathematical intersection between two weighted automata. States marked with 0 are starting states, and states marked with 1 are final states. Final automaton (right) accepts strings that both left automata accept. Final weights are obtained simply by summing up weights on common paths of the original two automata.

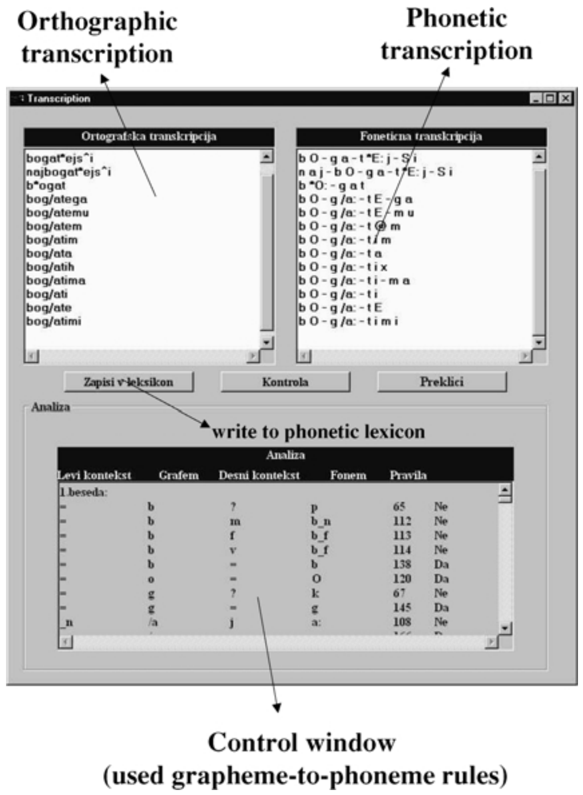
(also for $A_{\text{onsets}}$ and $A_{\text{codas}}$) can only be computed between automata because the accepting automata are closed under the intersection, and the finite-state transducers are not.

***5.2.8. Conclusions on the Syllabification Performance.*** Slovenian syllable phonotactics can be described by the same extended regular expression as in the cases of German and English. The onset and coda parts of the syllable are optional, whereas the nucleus part is obligatory and also defines the number of syllables in the word. The weights of the transitions in the syllabification transducer were obtained from training data derived directly from the frequencies of the onset, nucleus and coda types. They obviously play a significant role in obtaining the correct syllabification, especially in the case of consonant clusters. The approach presented describes a time and space optimal solution for syllabification and can be applied to any language. A syllabification transducer for the Slovenian language was constructed for illustration purposes. When the list of all syllables was obtained, the splitting of all syllables was performed, and the corresponding frequencies of onsets, nucleus and codas also were calculated. Frequencies were used in the construction of the syllabification weighted finite-state transducer. The final transducer for the Slovenian language consists of 4 states, where state 0 represents the starting state. States 2 and 3 represent the final states (Fig. 3). The constructed transducer was tested on the test database (every tenth entry in the SIflex phonetic lexicon −6,000, which were excluded from the training set). The syllabification transducer during testing did not produce any unacceptable syllabification.

### *5.3. Verification of Automatically Generated Phonetic Transcriptions*

To verify the automatically generated phonetic transcription, the expert must open the window that visually presents the phonetic transcription (performed with the automatic grapheme-to-phoneme conversion module). This window is shown in Fig. 5. The user is actually able to switch between using rule-based or data-driven (using a neural network) approaches. Only unique orthographic words are sent to the grapheme-to-phoneme transcription module since a lot of inflectional forms for corresponding items are the same.

The grapheme-to-phoneme transcription module receives only non-duplicated words as input and returns



| Ortografska transkripcija | Foneticna transkripcija |
|---|---|
| bogat*ejs^i | b O - g a - t *E: j - S i |
| najbogat*ejs^i | n a j - b O - g a - t *E: j - S i |
| b*ogat | b *O: - g a t |
| bog/atega | b O - g /a: - t E - g a |
| bog/atemu | b O - g /a: - t E - m u |
| bog/atem | b O - g /a: - t @ m |
| bog/atim | b O - g /a: - t i m |
| bog/ata | b O - g /a: - t a |
| bog/atih | b O - g /a: - t i x |
| bog/atima | b O - g /a: - t i - m a |
| bog/ati | b O - g /a: - t i |
| bog/ate | b O - g /a: - t E |
| bog/atimi | b O - g /a: - t i m i |

Analiza

| Levi kontekst | Grafem | Desni kontekst | Fonem | Pravila | |
|---|---|---|---|---|---|
| 1.beseda: | | | | | |
| - | b | ? | p | 65 | Ne |
| - | b | m | b_n | 112 | Ne |
| - | b | f | b_f | 113 | Ne |
| - | b | v | b_f | 114 | Ne |
| - | b | - | b | 138 | Da |
| - | o | - | O | 120 | Da |
| - | g | ? | k | 67 | Ne |
| - | g | - | g | 145 | Da |
| _n | /a | j | a: | 108 | Ne |

*Figure 5.* Grapheme-to-phoneme conversion window.

the corresponding phonetic transcriptions with syllable break marks '-' and stress marks. The linguistic expert verifies the results and submits everything into the phonetic lexicon. The control window in Fig. 5 can be used for inspection (of applied rules) to identify problems in the case of errors in the phonetic transcription obtained (when the rule-based approach is used).

### 5.4. *Notation and Attribute/Value Tables*

The notation format of the system's output for building morphological and phonetic lexicons consists of linear strings of characters representing the morphosyntactic information to be associated with word forms. The string was constructed by following the philosophy of the Intermediate Format proposed in the EAGLES Corpus proposal (Leech and Wilson, 1994). It consists of agreed upon symbols in predefined and fixed positions. The categories used, with their relevant attributes and values, are based on EAGLES documents (MULTEXT project, 1996).

## 6. Using the Finite-State Transducer Theory for Representation of Very Large Scale Lexicons

In general, when representing lexicons by automata, many entries share the same codes (strings, representing some piece of information). Thus, the number of codes is small compared to the number of entries. Newly developed lexicons are becoming more and more accurate, and the number of codes can increase considerably. This increase in the number of codes also increases the smallest possible size of such lexicons. Different codes need to be distinguished during the construction of the automaton. Therefore, the space required for an efficient hashing of the codes can also become costly. Since morphological and phonetic lexicons can be viewed as a list of string pairs, their representation using finite-state transducers seems to be very appropriate (Mohri, 1994, 2000, 2001). The results given at the end of this paper also confirm this assumption. Representation of lexicons using finite-state transducers, also provides reverse look-up capability.

### 6.1. *Compilation Process of Large Scale Lexicons into Finite-State Transducers*

***6.1.1. Lexicon Preparation.*** The methods used in the compilation of large scale lexicons into finite-state transducers (FST) assume that lexicons are given as large lists of strings and not as a set of rules as, for instance, was considered by Mehryar Mohri (1996). Morphological and phonetic lexicons can be viewed as a list of string pairs. Some items from German phonetic and morphological lexicons are shown in Fig. 6.

As with automata, direct construction of the sequential transducer representing a large-scale lexicon is impossible because the construction leads to a blow up for a large number of entries. The lexicon is split into several parts, to avoid this. Construction of the corresponding sequential transducers, including a minimization operation, then follows. When using union, determinization, and minimization operations, only one transducer representing the whole lexicon is obtained at the end (Fig. 7).

***6.1.2. Determinization of Finite-State Transducers.*** The algorithm used is close to the powerset construction used for the determinization of automata (Mohri, 1996). The main difference is that, in this case, states

| | |
|---|---|
| "Abte | "Abte |
| "E p - t @ | abt.mask(NS1,NP12)#0:amM:gmM:nmM |
| "Abten | "Abten |
| "E p - t @ n | abt.mask(NS1,NP12)#0:dmM |
| "Abtissin | "Abtissin |
| E p - t "I - s I n | "Abtissin.fem(NS0,NP5)#0:aeF:deF:geF:neF |
| "Abtissinnen | "Abtissinnen |
| E p - t "I - s I - n @ n | "Abtissin.fem(NS0,NP5)#0:amF:dmF:gmF:nmF |
| "Ackern | "Acker |
| "E - k 6 n | acker.mask(NS2,NP11)#0:amM:gmM:nmM |
| "Aderchen | "Aderchen |
| "E: - d 6 - C @ n | "Aderchen.neut(NS2,NP0)#0:aeN:amN:deN:dmN:gmN:neN |
| "Aderchens | "Aderchens |
| "E: - d 6 - C @ n s | "Aderchen.neut(NS2,NP0)#0:geN |
| ....... | ......... |
| (a) | (b) |

*Figure 6.* German phonetic (a) and morphological lexicons (b). German morphological lexicon is coded according to CISLEX specification (Guenthner and Maier, 1994).

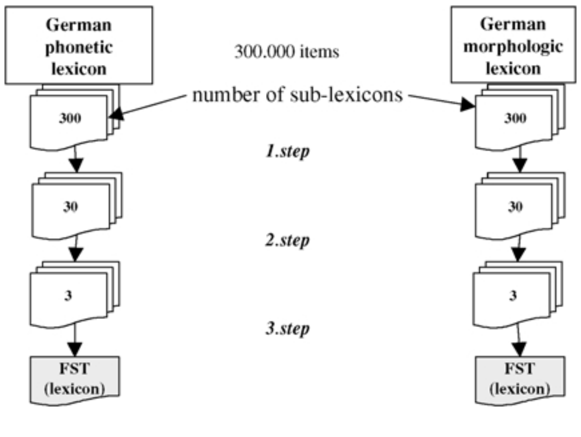


*Figure 7.* Lexicons preparation.

for the sets with strings need to be provided. These strings correspond to a delay in emission which is due to the fact that outputs corresponding to a given input can be different. Only the longest common prefix of outputs can be kept, and subsets represent actual pairs (state, string). The pseudocode for the algorithm to determinize a transducer $T_1$ is given in Fig. 8.

At each step a new state $q_2$ is considered, as can be seen in line 5 of Fig. 8. State $q_2$ is a final state only if it contains a pair $(q, w)$, where $q$ is final in $T_1$. String $w$ is the final output at the state $q_2$. In line 10, each input label $a$ of the transitions leaving the states of the subset $q_2$ is considered. A transition is constructed from state $q_2$ to state $\delta_2(q_2, a)$ with output $\sigma_2(q_2, a)$. Output $\sigma_2(q_2, a)$ represents the longest common prefix of the output labels of all the transitions leaving the states $q$ of $q_2$ with input label $a$, when left concatenated with their delayed string $w$. State $\delta_2(q_2, a)$ is the subset made of pairs $(q', w')$. Here $q'$ is a state reached by one of the transitions with input label $a$ in $T_1$, and $w' = [\sigma_2(q_2, a)]^{-1} w \sigma_1(q, a, q')$ is the delayed string that could not be outputted earlier in the algorithm. String $[\sigma_2(q_2, a)]^{-1} w \sigma_1(q, a, q')$ is a well defined string, because $[\sigma_2(q_2, a)]$ is a prefix of all $w\sigma_1(q, a, q')$, as can be seen from line 10.

In Fig. 10 the result of using the determinization algorithm on the transducer from Fig. 9 (obtained using the union operation) is shown. In this example the number of states of the determinized transducer $T_2$ is already less than in $T_1$.

Experiments show that this method is very efficient in constructing transducers for representation of large lexicons. The disadvantage of this algorithm is that the outputs are pushed toward the final states, which creates a long delay in emission. Fortunately, sequential transducers can be minimized as shown in the next section. An important characteristic of the minimization algorithm is that it pushes back outputs as much as possible

```
    Determinize_transducer( T_1,T_2)
1   $F_2 \leftarrow \emptyset$
2   $i_2 \leftarrow \bigcup_{i \in I_1} \{(i, \varepsilon)\}$
3   $Q_2 \leftarrow \{ i_2 \}$
4   while $Q \neq \emptyset$
5    do $q_2 \leftarrow$ head[Q]
6     if (there exists $(q, w) \in q_2$ such that $q \in F_1$ )
7         then $F_2 \leftarrow F_2 \cup \{q_2\}$
8              $\phi_2(q_2) \leftarrow w$
9   for each $a$ such that $(q,w) \in q_2$ and $\delta_1(q,a)$ defined do
10      $\sigma_2(q_2, a) \leftarrow \bigwedge_{(q,a) \in J_1(a)} \left[ w \cdot \bigwedge_{q' \in \delta_1(q,w)} \sigma_1(q, a, q') \right]$
11      $\delta_2(q_2, a) \leftarrow \bigcup_{(q,w,q') \in J_2(a)} \{(q', [\sigma_2(q_2, a)]^{-1} w \cdot \sigma_1(q, a, q'))\}$
12   if $(\delta_2(q_2, a)$ is a new state)
13      then Enqueue(Q,$\delta_2(q_2, a)$)
14  Dequeue(Q)
```

*Figure 8.* Pseudocode for determinization algorithm (Mohri, 1996).

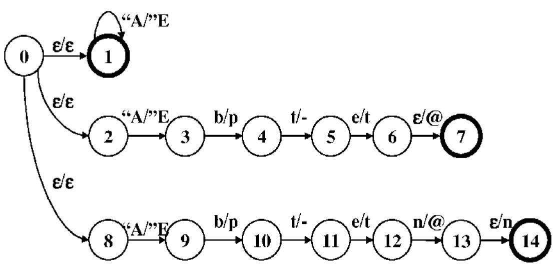


*Figure 9.* Union operation done on a few word items in the German phonetic lexicon ($T_1$).

toward the initial state. In such a way the problem just mentioned can be eliminated.

***6.1.3. Minimization of Finite-State Transducers.*** Sequential transducers allow very fast look-up. Minimization algorithms also help to make them space efficient (Watson, 1993, 1995; Mohri, 1996, 2000). The whole minimization procedure for sequential transducers actually consists of two different algorithms. One

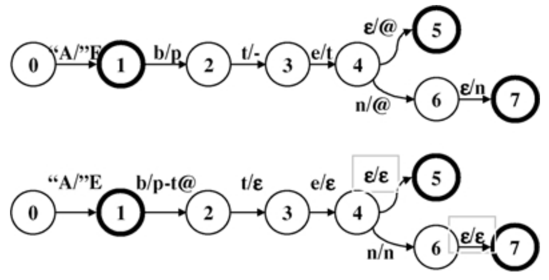


*Figure 10.* Finite-state transducers $T_2$ (top) and $T_3$ (bottom) obtained after performing determinization and prefixation algorithms on the finite-state transducer shown in Fig. 9.

is an algorithm for computing the prefix of a non-deterministic automaton (Mohri, 1994), and the other is a classical algorithm for the minimization of automata (Bauer, 1988; Watson, 1993, 1995). This section presents an algorithm for the computation of the prefix, as it is independent of the sequential transducer concept, and will describe the entire algorithm that allows the derivation of minimal sequential transducers.

The following notations are used in the algorithm described:

- $G^T$ the transpose of $G$ (the automaton obtained from $G$ by reversing each transition);
- *Trans*$[u]$ the set of transitions leaving $u \in V$;
- *Trans*$^T[u]$ the set of transitions entering $u \in V$;
- $t.v$ the vertex reached by $t$ and $t.l$, its label, for any transition $t$ in *Trans*$[u]$ (resp. in *Trans*$^T[u]$), $u \in V$;
- *out-degree*$[u]$ the number of edges leaving $u \in V$;
- *in-degree*$[u]$ the number of edges entering $u \in V$;
- $E$ the set of edges of $G$.

1. First, $\pi_u$, the greatest common prefixes of all its leaving transitions, are computed:

$$\pi_u \leftarrow \left( \bigwedge_{\substack{t \in \text{Trans}[u] \\ t.v \in \text{scc}}} t.l\, X_{t.v} \right) \wedge \left( \bigwedge_{\substack{t \in \text{Trans}[u] \\ t.v \notin \text{scc}}} t.l \right) \quad \text{if } u \notin F,$$

$$\pi_u \leftarrow \varepsilon \quad \text{else;}$$

2. Then if $\pi_u \neq \varepsilon$, a change of variables can be made: $Y_u \leftarrow \pi_u X_u$. This second step is equivalent to storing the value $\pi_u$ and solving the system modified by the following operations:

$$\forall t \in \text{Trans}[u], \quad t.l \leftarrow \pi_u^{-1} t.l,$$
$$\forall t \in \text{Trans}^T[u], \quad t.l \leftarrow t.l \pi_u.$$

The number of times these two operations are performed can be limited by storing in array $N$ the number of empty labels leaving each state $u$ of the strongly connected component *scc*. While $N[u] \neq 0$, there is no use performing these operations as the value of $\pi_u$ is $\varepsilon$. Also in the case that $N[u] = 0$ right after the computation of $\pi_u$, the $\pi_u$ will remain equal to $\varepsilon$, as changes of variables will only affect suffixes of the transitions leaving $u$. This information can be stored using an array $F$, in order to avoid performing step 1 in such situations or when $u$ is a final state. In the algorithm a queue $Q$ is used, containing the set of states $u$ with $N[u] = F[u] = 0$, for which the two operations above need to be performed, and an array *INQ* indicating for each state $u$ whether it is in queue $Q$.

The above operations are started by initializing $N$ and $F$ to 0 for all states in *scc*, and by enqueuing in queue $Q$ an arbitrarily chosen state $u$ of the strongly connected component *scc*. Each time the transition of a state $v$ of $Trans^T[u]$ is modified, $v$ is added to $Q$ if $N[v] = F[u] = 0$. The property of scc's (strongly connected component) and the initialization of $N$ and $F$ assure that each state of *scc* will be enqueued at least once. Steps 1 and 2 are performed until queue $Q = \emptyset$. This must happen as, except for the first time, step 1 is performed for a state $u$ if $N[u] = 0$. After the computation of the greatest common prefix, $N[u] = 0$ is obtained, and then $u$ will never be enqueued again, or $N[u] \neq 0$ and then a new non-empty factor $\pi_u$ of $P(u)$ has been identified. It is obvious, then, that each state $u$ is enqueued at most $(|P(u)| + 2)$ times in $Q$, and after, at most, $(|\text{Pmax}| + 2)$ steps, $Q = \emptyset$ is obtained.

Once $Q = \emptyset$, it is easy to see that the system of equations has a trivial solution: $\forall u \in scc$, $X_u = \varepsilon$. It has a unique solution. Therefore, the system is resolved. Concatenating the factors $\pi_u$ involved in the changes of variables corresponding to the state $u$ gives the value of $P(u)$. The set of operations (2) are obviously equivalent to multiplying the label of each transition joining the states $u$ and $v$, ($v \in scc$), on the right by $P(v)$ and on the left by $[P(u)]^{-1}$ if $u$ is in *scc*. Thus the transformations described above do modify the transitions leaving or entering states of *scc* as desired. The pseudocode described in Fig. 11 gives an algorithm that computes $p(G)$ from $G$. In the algorithm, $V(G^{SCC})$ represents the set of states of the component graph of $G$. For each $u$ in $V(G^{SCC})$, $SCC[u]$ stands for the strongly connected component corresponding to $u$. The function *GCP*$(G, u)$ called in the algorithm is such that it

```
***Prefix_Computation(G)***
1   for each u ∈ V(G^SCC)
2   do for each v ∈ SCC[u]
3     do N[v] ← INQ[v] ← F[v] ← 0
4   Q ← v
5   INQ[v] ← 1
6   while Q ≠ ∅
7     do v ← head[Q]
8        Dequeue(Q)
9        INQ[v] ← 0
10       p ← GCP(G,v)
11       for each t ∈ Trans^T[v]
12         do if(p ≠ ε)
13              then if(t.v ∈ SCC[v] and N[t.v] > 0
                              and t.l = ε and F[t.v] = 0)
14                     then N[t.v] ← N[t.v] – 1
15                     t.l ← t.l p
16              if(N[t.v] = 0 and INQ[t.v] = 0 and
                               F[t.v] = 0)
17                     then Enqueue(Q,t.v)
18                     INQ[t.v] = 1
```

*Figure 11.* Pseudocode for the prefixation algorithm on finite-state transducers (Mohri, 2000).

returns $p$, which is the greatest common prefix of all transitions leaving $u$ ($p = \varepsilon$ if $u \in F$). It replaces each of these transitions by dividing them on the left by $p$ and counts and stores in $N[u]$ the number of empty transitions. If $N[u] = 0$ after the computation of the greatest common prefix, or if $u$ is a final state, the $F[u]$ becomes 1.

The computation of the greatest common prefix of $n(n > 1)$ words requires at most $(|p| + 1) \cdot (n - 1)$ comparisons, where $p$ is the result of this computation (Mohri, 2000). This operation consists of comparing the letters of the first word to those of the $(n - 1)$ others until a mismatch or end of a word occurs. The same comparisons allow obtaining the division on the left by $p$ and the number of empty transitions. In case only one transition leaves $v$, the computation of the greatest common prefix can be assumed to be in $O(1)$. Therefore, the

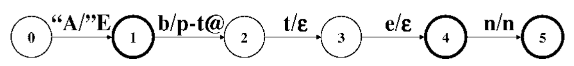


*Figure 12.* Finite-state transducer $T_4$ obtained using minimization algorithm in the sense of automata from $T_3$.

cost of a call of the function *GCP* for a state $v(\in V - F)$ is $O((|p| + 1)\ (out\text{-}degree(v) - 1) + 1)$. Here $p$ is the greatest common prefix of the transitions leaving $v$.

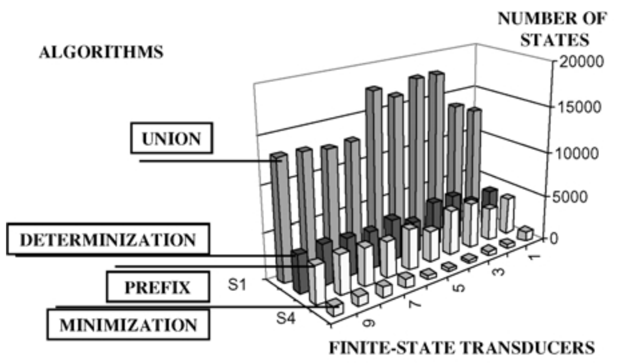
Given a sequential transducer $T$, the application of the *prefix computation* algorithm (Mohri, 2000) to the output automaton of $T$ has no effect on the states of $T$ or on its transition function. Only the output function $\sigma$ of T changes. A minimal *ST*, that computes the same function as $T$, can be obtained by applying the *prefix computation* algorithm to the output automaton of $T$, and also the minimization algorithm, in the sense of automata, to the resulting transducer (Watson, 1993, 1995). Figure 10 (transducer $T_3$) shows the result obtained after performing a prefix computation algorithm on the sequential transducer $T_2$ in this particular case. The application of the *prefix computation* algorithm on $T_2$ leads to the transducer $T_3$, which computes the same function. Only outputs differ from those of $T_2$. In Fig. 12 the final obtained transducer is presented using a minimization algorithm in the sense of automata.

## 7. Results

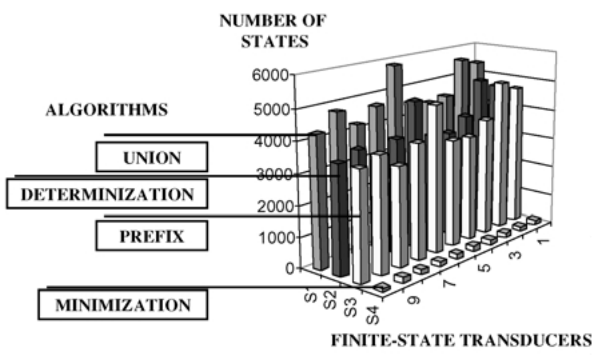
The German (CISLEX (Guenthner and Maier, 1994)) and the Slovenian large scale phonetic and morphological lexicons were used for the lexicons' compilations. In the compilation process a large set of proprietary programs written in C++ were used that perform many operations efficiently on finite-state transducers and finite-state automata, including determinization, minimization, union, intersection, compaction, prefixation, and local extension.

The following algorithms were used during the construction of corresponding finite-state transducers: union, determinization, prefix computation, and classical minimization algorithms of finite-state automaton (Hopcroft et al., 1979; Watson, 1993, 1995; Mohri, 2001). A prefix computation algorithm was applied before the minimization algorithms. It pushes the output labels towards the initial state as much as possible.

In the first step of compilation, a significant reduction in the number of states was achieved, which is evident from Fig. 13. This is another reason for following the procedure described under Section 7.3.3 (Fig. 10). The number of states has already decreased as expected after application of the determinization algorithm. The number of states obviously remain unchanged after performing the *prefix computation* algorithm. This algorithm works only on the output automaton of the corresponding transducer. It pushes back outputs as much as possible toward the initial state. The effect of the *prefix computation* algorithm can only be noticed at the end of the compilation process, when much smaller finite-state transducers are obtained than in the case where only a classical minimization algorithm would be performed after determinization. The explanation for this can be found in Fig. 10 (transducers $T_2$ and



(a)



(b)

*Figure 13.* Achieved reduction in the number of states obtained during the first step of compilation process—10 randomly chosen transducers (a) German phonetic lexicon. (b) German morphology lexicon.

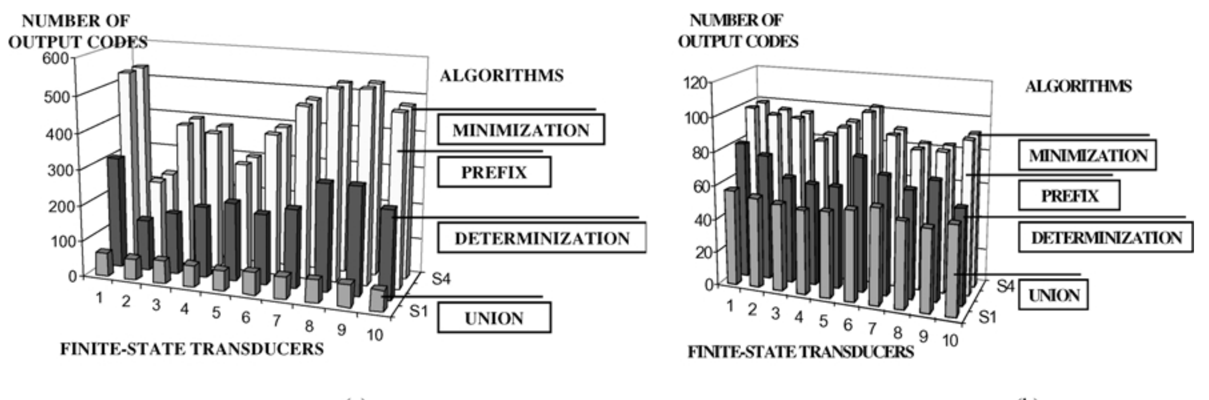


*Figure 14.* Increasing number of output codes in the first step of compilation process—10 randomly chosen transducers (a) German phonetic lexicon. (b) German morphological lexicon.

$T_3$). In the transducer $T_3$ there are, after performing the *prefix computation* algorithm, newly created $\varepsilon/\varepsilon$ transition labels. These empty transition labels are the result of pushing back outputs towards the initial state. This is why much smaller transducers are obtained after performing the minimization algorithm in the sense of automata.

As shown in Fig. 14, the number of output codes increased after *determinization* and the *prefix computation* algorithm were performed. In those cases where the *prefix computation* is not performed, the final number of codes would be significantly smaller, but the final transducer would be much bigger (more states and transitions). According to the experiments, it only makes sense to have more codes and a much smaller transducer. It is also interesting that in the compilation of the morphological lexicon, many more output codes are generated as in the case of the phonetic lexicon (Fig. 14). The reason is that the morphological lexicon is comprised of much more information than the phonetic lexicon. In the second step of the compilation process, the same situation regarding state reduction can be observed as in the first step (Fig. 15). Only the reduction in the number of states is smaller, and there is no significant increase in the number of output codes (Fig. 16). In Table 5 the final results are given for the obtained finite-state transducers for the German phonetic and morphological lexicons. The number of input codes is the same for both lexicons, and the number of output codes is twice as big in the case of the

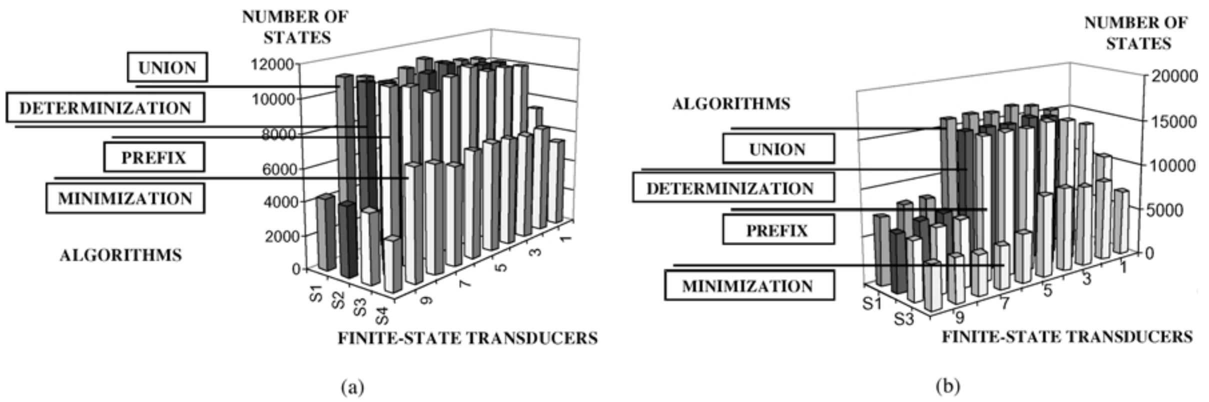


*Figure 15.* Achieved reduction of the number of states obtained in the second step of compilation process—10 randomly chosen transducers (a) German phonetic lexicon. (b) German morphological lexicon.

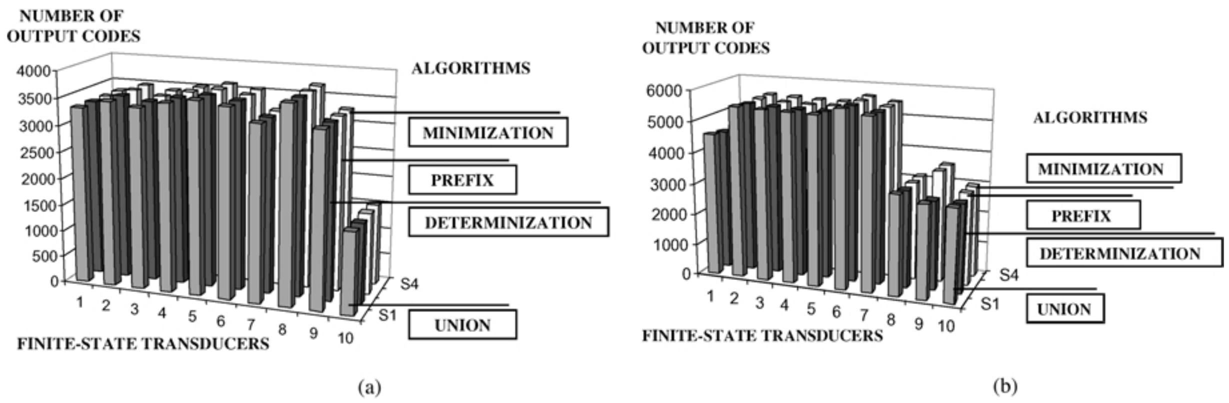


*Figure 16.* Increasing number of output codes in the second step of compilation process—10 randomly chosen transducers (a) German phonetic lexicon. (b) German morphological lexicon.

morphological lexicon. The reason is that the information field in the morphological lexicon is substantially longer (Fig. 6).

The same representation using finite-state transducers was also performed on the SIflex and SImlex Slovenian lexicons. The starting size was 1.8 MB (60,000 items) for SIflex and 1.4 MB for SImlex (40,000 items). The final size achieved using the presented algorithms was 352 KB for SIflex and 662 KB for SImlex.

*Table 5.* The final finite-state transducers representing German phonetic ($FST_1$) and German morphological ($FST_2$) lexicons (300,000 items).

| | $FST_1$ | $FST_2$ |
|---|---|---|
| Number of input codes | 61 | 61 |
| Number of output codes | 34,879 | 87,204 |
| Size of output vocabulary | 343 KB | 3,6 MB |
| Number of states | 112,498 | 169,613 |
| Number of transitions | 200,801 | 325,839 |
| Size of ASCII file | 6,6 MB | 11,53 MB |
| Size of bin file | 2,78 MB | 6,33 MB |

*Table 6.* The final finite-state transducers representing Slovenian phonetic ($FST_1$) (60,000 items) and Slovenian morphological lexicons ($FST_2$) (40,000 items).

| | $FST_1$ | $FST_2$ |
|---|---|---|
| Number of states | 69,498 | 90,613 |
| Number of transitions | 90,801 | 130,839 |
| Size of bin file | 252 KB | 662 KB |

## 8. Conclusions

Currently, six linguistic experts are working intensively with this system, in order to build Slovenian morphological and phonetic lexicons. Following more than a year of intensive work, the system was evaluated as a very efficient tool for the expert. The linguistic experts find it very easy to use, accurate enough in automatic generation of linguistic descriptions for items and also in grapheme-to-phoneme transcriptions. Errors can be corrected quickly and easily without extensive typing, mostly using a mouse. They are able to verify approximately 100 root items in 15 hours. Because the Slovenian language is a very inflectional language, on average 30 inflectional forms per root item are generated (during the analysis of verbs, up to 200 inflectional forms can be generated). This means that, on average, 3000 inflectional forms for 100 root items are achieved. The phonetic lexicon is, on average, ten times smaller than the morphologic lexicon, since many duplicated inflectional forms are obtained during conjugation/declension. The SImlex morphologic lexicon currently contains more than 600,000 items (root items plus corresponding inflectional forms) and a phonetic lexicon that is approximately ten times smaller (60,000 items). It is planned to use the Slovenian phonetic lexicon for research work in the field of automatic continuous speech recognition for the Slovenian language. Both phonetic and morphological lexicons will be used for Slovenian TTS synthesis.

Minimal memory usage and fast look-up times are desired when using lexicons in run-time systems. Lexicons can be quite huge; it is, therefore, very important that their representation be optimal. As shown in this paper, the representation of lexicons using finite-state transducers fulfil's both requirements. They provide fast look-up time, double side look-up, and compactness.

## References

Aho, A.V., Hopcroft, J.E., and Ullman, J.D. (1974). *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison Wesley.

Bauer, W. (1988). On minimizing finite automata. *EATCS Bulletin*, 35.

Berstel, J. and Reutenauer, C. (1988). *Rational Series and Their Languages*. Berlin, New York: Springer-Verlag.

Crochemore, M. (1986). Transducers and repetitions. *Theoretical Computer Science*, 45.

Guenthner, F. and Maier, P. (1994). Das CISLEX Woerterbuch system, CIS-Bericht-94-76.

Hopcroft, J.E. and Ullman, J.D. (1979). *Introduction to Automata Theory, Languages, and Computation*. Reading, MA: Addison Wesley.

Kiraz, G.A. and Möbius, B. (1998). Multilingual syllabification using weighted finite-state transducers. *Bell Labs—Lucent Technologies*.

Kuich, W. and Arto, S. (1986). Semirings, automata, languages. Number 5 in *EATCS Monographs on Theoretical Computer Science*. Berlin, Germany: Springer Verlag.

Leech, G.N. and Wilson, A. (1994). Morphosynthactic Annotation. EAGLES Bericht EAG-CSG/IR-T3.1. Instituto di Linguistica Computazionale, Pisa.

Mohri, M. (1994). Minimization of sequential transducers. *Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching*, Berlin: Springer Verlag, pp. 151–163.

Mohri, M. (1996). On some applications of finite-state automata theory to natural language processing. *Natural Language Engineering 1*, Cambridge University Press.

Mohri, M. (2000). Minimization algorithms for sequential transducers. *Theoretical Computer Science*, *234*:177–201.

Mohri, M. (2001). Language processing with weighted transducers. *Proceedings of the 8th Annual Traitement Automatique des Langues Naturelles (TALN 2001)*. Tours, France.

MULTEXT project lexical specifications (1996). http://www.lpl.univaix.fr/projects/multext/LEX/LEX.Specifications.html.

Rojc, M. and Kačič, Z. (2000). A computational platform for development of morphologic and phonetic lexica. *Second International Conference on Language Resources and Evaluation*. Athens, Greece.

Rojc, M., Stergar, J., Ralph, W., Hain, U., Holzapfel, M., and Horvat, B. (1999). A multilingual text processing engine for text-to-speech synthesis system. *Proceedings EUROSPEECH 1999*, Budapest, pp. 2107–2110.

Watson, B. (1993). A taxonomy of finite automata minimization algorithms. Computing Science Report 93/44, Eindhoven University of Technology, The Nederlands.

Watson, B. (1995). Taxonomies and toolkits of regular language algorithms. Ph.D. Thesis, Eindhoven University of Technology and Computing Science.

Zell, A. (1994). *Simulation Neuronaler Netze*. Addison-Wesley.