Using Finite-State Transducer Theory for Representation of Very Large Scale Lexicons

Matej Rojc, Zdravko Kačič
Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova 17, Maribor, Slovenia
Phone: +386 2 220 7223, Fax: +386 2 2511 178
matej.rojc@uni-mb.si, kacic@uni-mb.si

Keywords: finite-state transducers, natural language resources, multilingual text-to-speech synthesis, morphology, lexicons

Received: September 17, 2002

In multilingual text-to-speech synthesis systems, many external extensive natural language resources are used, especially in the text processing part. Therefore it is very important 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 represents an important problem because of the needed space and slow lookup-time. In the paper a method and results of compiling large lexicons, with an example of compiling German phonetic and morphology lexicons (CISLEX), into corresponding finite-state transducers (FSTs) are presented. Each lexicon consisted of about 300.000 words. Representation of large lexicons using finite-state transducers is mainly motivated by considerations of space and time efficiency. For both lexicons a great reduction in size and optimal access time was achieved. The starting size for German phonetic lexicon was 12.53 MB and 18.49 MB for morphology lexicon. The final size of the corresponding FST was only 2.78 MB for the phonetic lexicon and 6.33 MB for the morphology lexicon. At the same time the look-up time is optimal, since it depends only on the length of the input word and not on the size of the lexicon. Using such representation, the integration of lexicons for new languages into the multilingual TTS system is easy and does not require any changes of algorithms that use such lexicons.

1 Motivation

Finite-state machines are already used in many areas of natural language processing. From the computational point of view, their use is mainly motivated by considerations of space and time efficiency. Linguistically, the finite-state machines [6][8][10][11] allow one to describe easily most of the relevant local phenomena in the language. They provide also compact representation of external language specific resources needed for knowledge representation in the automatic text-to-speech synthesis systems. These features of finite-state machines are of major importance especially when we are dealing with multilingual text processing in text-to-speech synthesis systems (TTS systems).

In multilingual text-processing module for the multilingual TTS system, external natural language resources (e.g., phonetic, morphology lexicons etc.) represent an important problem, regarding the memory usage and time needed for lookup process.

In the following sections we are presenting an approach for compiling such lexicons into finite-state transducers that represent their time and space optimal representation. The effect of using finite-state transducers for representation of external natural language resources is great reduction of the memory usage required by the lexicons and the optimal access time (required for obtaining information) that is independent from the size of the lexicons. The whole compilation process into finite-state transducers will be presented and at the end results obtained for the German lexicons described.

2 Finite-state automata and finite-state transducers

2.1 Finite-state automata (FSA)

Finite-state automata (FSA) [6] 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.

Definition

A finite-state automaton A is a 5-tuple (Σ, Q, i, F, E) where Σ 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 $E \subseteq Qx(\Sigma \cup \{\in\})xQ$ 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.

2.2 Finite-state transducer (FST)

FSTs [9] can be interpreted as defining a class of graphs, a class of relations on strings, or a class of transductions on strings. On the first interpretation, an FST can be seen as an FSA, in which each arc is labelled by a pair of symbols rather than by a single symbol.

Definition
A finite-state transducer T is a 6-tuple ($\Sigma_1, \Sigma_2, Q, i, F, E$) such that:

- Σ_1 is a finite alphabet, namely the input alphabet,
- Σ_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,
- $E \subseteq Qx\Sigma_1^* x\Sigma_2^* xQ$ is the set of edges.

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

2.3 Use of FSMs for time and space optimal Lexicon representation

When representing lexicons by automata, in general, many entries share the same codes (strings, representing some piece of information). The number of codes is then small compared to the number of entries. Newly developed lexicons are more and more accurate and the number of codes can increase considerably. The increase in number of codes also increases the smallest possible size of such lexicons. During the construction of the automaton one needs to distinguish different codes, therefore space required for an efficient hashing of the codes can also become costly.

Available lexicons that were used in this experiment suggest that the representation by automata would be less appropriate. Since morphological and phonetic lexicons can be viewed as a list of pairs of strings, their representation using finite-state transducers [10] seems to be very appropriate. The results given at the end of this paper also confirm this assumption. Representation of lexicons using finite-state transducers on the other hand also provides reverse look-up capability.

In the multilingual TTS system morphological and phonetic lexicons represent part of the external natural language dependent resources used by multilingual text-processing engine. It is desired that language independent modules for morphology analysis and grapheme-to-phoneme conversion inside the multilingual text-processing engine use common algorithms for multiple languages. This is possible when external natural language dependent resources are represented as finite-state transducers. Integration of new lexicons for new languages in the whole TTS system is then very easy, since only compilation procedure (off-line) has to be performed.

3 Compilation process of large scale lexicons into finite-state transducers

3.1 Lexicons preparation

The methods used in the compilation of large scale lexicons into finite-state transducers (FST) assume that the lexicons are given as large lists of strings and not as a set of rules as considered by Mehryar Mohri [3] for instance. Obviously morphological and phonetic lexicons can be viewed as a list of pairs of strings and their representation using finite-state transducers seems to be very appropriate. In Fig. 1 some items from German phonetic and morphology lexicons are shown.

As with automata, direct construction of the sequential transducer representing a large-scale lexicon, is not possible because the construction leads to a blow up for a large number of entries. To avoid this, splitting the lexicon into several parts is performed. Then the construction of the corresponding sequential transducers including minimization operation follows. Using union, determinization, and minimization operations, only one transducer representing the whole lexicon is obtained at the end (Fig.2).

3.2 Determinization of finite-state transducers

The algorithm used is close to the powerset construction used for determinizing automata [3]. The main difference is that here one needs to provide states of the sets with strings. These strings correspond to a delay in the emission that is due to the fact that outputs corresponding to a given input can be different. Therefore only the longest common prefix of outputs can be kept and subsets represent actually pairs (state, string). The pseudo-code for the algorithm to determinize a transducer T_1 is given in Fig. 3.

```
"Abte
"E p - t @
"Abten
"E p - t @ n
"Abtissin
E p - t "I - s I n
"Abtissinnen
E p - t "I - s I - n @ n
```

"Ackern
"E - k 6 n
"Aderchen
"E: - d 6 - C @ n
"Aderchens
"E: - d 6 - C @ n s
....

a)

"Abte
abt.mask(NS1,NP12)#0:amM:gmM:nmM
"Abten
abt.mask(NS1,NP12)#0:dmM
"Abtissin
"Abtissin.fem(NS0,NP5)#0:aeF:deF:geF:neF
"Abtissinnen
"Abtissin.fem(NS0,NP5)#0:amF:dmF:gmF:nmF
"Acker
acker.mask(NS2,NP11)#0:amM:gmM:nmM
"Aderchen
"Aderchen.neut(NS2,NP0)#0:aeN:amN:deN:dmN:gmN:neN:nmN
"Aderchens
"Aderchen.neut(NS2,NP0)#0:geN
....

b)

Figure 1: German phonetic (a) and morphology lexicons (b). German morphology lexicon is coded according to CISLEX specification [5].

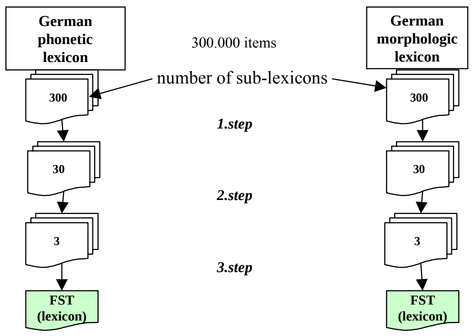


Figure 2: Lexicons preparation.

At each step a new state q_2 is considered as can be seen in line 5. 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 outputed earlier in the algorithm. String $[\sigma_2(q_2,a)]^{-1}w\sigma_1(q,a,q')$ is a well defined string since $[\sigma_2(q_2,a)]$ is a prefix of all $w\sigma_1(q,a,q')$ as can be seen from line 10.

In Fig. 5 we can see the result of using the determinization algorithm on transducer from Fig. 4 (obtained using union operation). In this example the number of states of the determinized transducer T_2 is already less than in T_1. Experiments showed 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 final states, which creates a long delay in emission. But fortunately sequential transducers can be minimized as we will show in the next section. An important characteristic of the minimization algorithm is that it pushes back outputs as much as possible toward the initial state. In such a way we can eliminate the problem just mentioned.

```
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 3: Pseudocode for determinization algorithm [3].

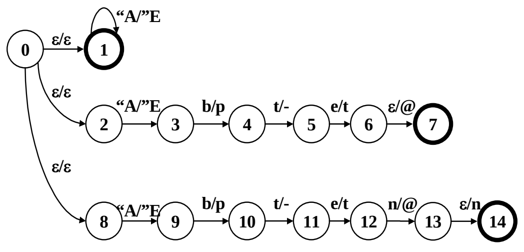


Figure 4: Union operation done on a few word items in the German phonetic lexicon (T_1).

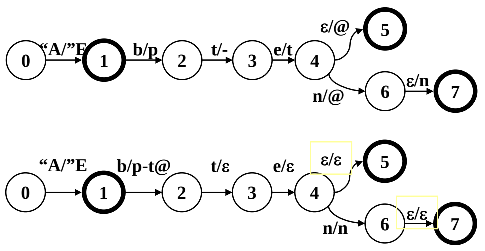


Figure 5: Finite-state transducers T_2 (above) and T_3 (below) obtained after performing determinization and prefixation algorithms on finite-state transducer showed in Fig. 4.

3.3 Minimization of finite-state transducers

Sequential transducers allow very fast look-up. But transducers can also be minimized. Minimization algorithms help to make them also space efficient [1][2][4][7]. The whole minimization procedure for sequential transducers consists actually of two different algorithms. One is algorithm for computation of the prefix of a non-deterministic automaton [4] and the other is classical algorithm for minimization of automata [1][2]. In this section we will present the algorithm for computation of the prefix, as it is independent of the concept of sequential transducers and will describe the entire algorithm that allows derivation of minimal sequential transducers.

In the algorithm described, we use the following notation:

- 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 we compute π_u, the greatest common prefixes of all its leaving transitions:

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

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

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

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 to perform these operations as the value of π_u is ε. Also in the case that *N[u] = 0* right after the computation of π_u, the π_u will remain equal to ε, 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 we use a queue *Q* 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 enqued at least once. *Steps 1* and *2* are operated until queue *Q* = $\varnothing$. 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 we can have *N[u] = 0* and then *u* will never be enqueued again, or *N[u]* $\neq$ *0* and then a new non empty factor π_u of *P(u)* has been identified. It is obviously then, that each state *u* is enqueued at most *(|P(u)|+2)* times in *Q*, and after at most *(|Pmax|+2)* steps we have *Q* = $\varnothing$.

Prefix_Computation(G)

```
for each $u \in V(G^{SCC})$
do for each $v \in SCC[u]$
   do $N[v] \leftarrow INQ[v] \leftarrow F[v] \leftarrow 0$
$Q \leftarrow v$
$INQ[v] \leftarrow 1$
while $Q \neq \emptyset$
   do $v \leftarrow head[Q]$
      Dequeue(Q)
      $INQ[v] \leftarrow 0$
      $p \leftarrow GCP(G,v)$
      for each $t \in Trans^T[v]$
         do if($p \neq \varepsilon$)
            then if($t.v \in SCC[v]$ and $N[t.v] > 0$
                 and $t.l = \varepsilon$ and $F[t.v] = 0$)
                 then $N[t.v] \leftarrow N[t.v] - 1$
                 $t.l \leftarrow t.l\ p$
            if($N[t.v] = 0$ and $INQ[t.v] = 0$ and
                 $F[t.v] = 0$)
                 then Enqueue(Q,t.v)
                 $INQ[t.v] = 1$
```

Figure 6: Pseudocode for the prefixation algorithm on finite-state transducers [4].

Once $Q = \varnothing$, 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 π_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, $(v \in scc)$, at right by $P(v)$ and at 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 above pseudocode 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 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 at 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 the value 1.

The computation of the greatest common prefix of n $(n>1)$ words requires at most $(|p|+1).(n-1)$ comparisons, where p is the result of this computation [4]. 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 to obtain the division at 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 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.

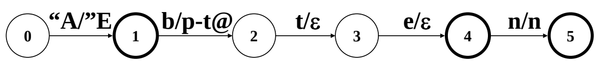
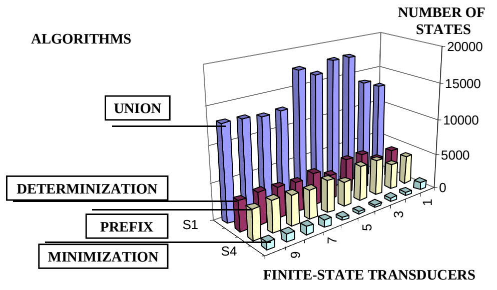


Figure 7: Finite-state transducer T_4 obtained using minimization algorithm in the sense of automata from T_3.

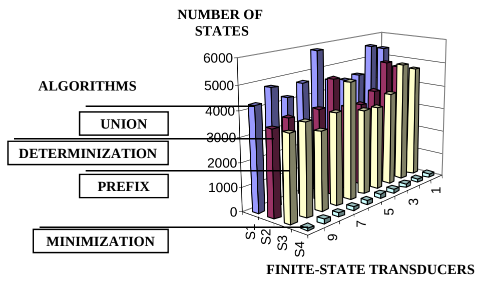
Given a sequential transducer T, the application of the *prefix computation* algorithm [4] to the output automaton of T has no effect on the states of T or on its transition function. Only the output function σ of T is changed. A minimal ST, that computes the same function as T, can be obtained by applying the *prefix computation* algorithm to the output automaton of T, and also the minimization algorithm in the sense of automata, to the resulting transducer [1][2]. Fig. 5 (transducer T_3) shows the result obtained after performing prefix computation algorithm on sequential transducer T_2 in 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. 7 the final obtained transducer is presented using minimization algorithm in the sense of automata.

4 Results

For the lexicons compilation the German large scale phonetic and morphology lexicons (CISLEX) [5] were used. In compilation process a large set of proprietary programs written in C++ that perform efficiently many operations on finite-state transducers and finite-state automata including determinization, minimization, union, intersection, compaction, prefixation, local extension and others were used.



a)



b)

Figure 8: Achieved reduction of the number of states obtained in the first step of compilation process – 10 randomly chosen transducers (*a:* phonetic lexicon. *b:* morphology lexicon.)

During construction of corresponding finite-state transducers, the following algorithms were used: union, determinization, prefix computation and classical minimization algorithms of finite-state automaton (Aho, Sethi, and Ullman; Hopcroft; Watson) [1][2][12]. Prefix computation algorithm was used before minimization algorithms. It pushes the output labels towards the initial state as much as possible.

All lexicons represent part of the language dependent external knowledge for morphology and grapheme-to-phoneme modules in the multilingual text-to-speech processing system. The starting sizes of phonetic lexicon and morphology lexicon were 12.52 MB and 18.49 MB. Both lexicons contained 300.000 items. Final size of corresponding finite-state transducer was 2.78 MB (120.386 states) for the first one and 6.33 MB (183.123 states) for the second.

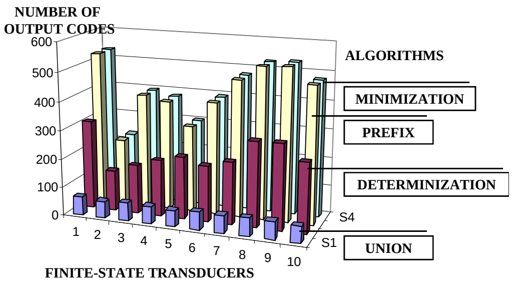
In the first step of compilation, the great reduction of the number of states was achieved, what is evident from Fig. 8. This is also the reason, why we follow the procedure described under subsection 2.2 (Fig. 2). The number of states decreased already after determinization algorithm as expected. The number of states obviously does not change after performing *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 *prefix computation* algorithm can be noticed only at the end of the compilation process, when much smaller finite-state transducers are obtained than in the case when only classical minimization algorithm after determinization would be performed. The answer for that can be found from the Fig. 5 (transducers T_2 and T_3*).* In the transducer T_3 we have after performing *prefix computation* algorithm newly created ε/ε transition labels. This empty transition labels are result of pushing back outputs toward the initial state. That's why after performing minimization algorithm in the sense of automata much smaller transducers are obtained.

In the Fig.9 we see that the number of output codes has increased after determinization and *prefix computation* algorithm were performed. In case that the *prefix computation* would not be performed, final number of codes would be significantly smaller, but the final transducer would also be much bigger (more states and transitions). According to the experiments it only makes sense to have more codes and much smaller transducer.

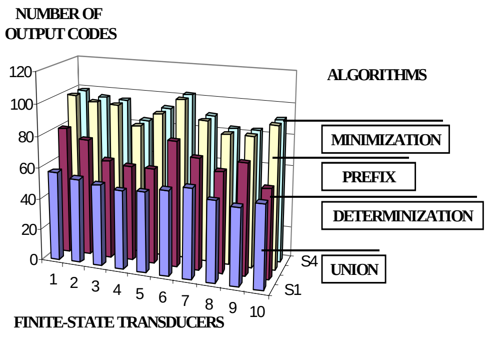
It is also interesting that in compilation of morphology lexicon, much more output codes is generated as in the case of phonetic lexicon (Fig. 9). The reason is that the morphology lexicon comprises much more information than the phonetic lexicon.

In the second step of the compilation process, the same situation regarding the state reduction can be observed as in the first step (Fig. 10). Only reduction of the number of states is smaller and there is no significant increase of the number of output codes (Fig. 11).

In Table 5 the final results for the obtained finite-state transducers for German phonetic and morphology lexicons are given. The number of input codes is the same for both lexicons and the number of output codes is two-times bigger in case of morphology lexicon. The reason is that the information field in the morphology lexicon is substantial longer (Fig. 1).

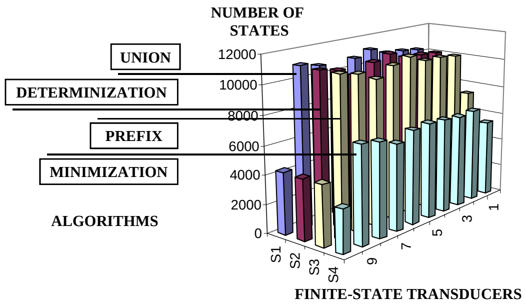


a)

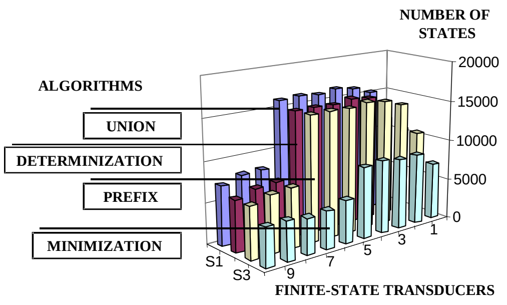


b)

Figure 9: Increasing number of output codes in the first step of compilation process – 10 randomly chosen transducers (*a:* phonetic lexicon. *b:* morphology lexicon.)



a)

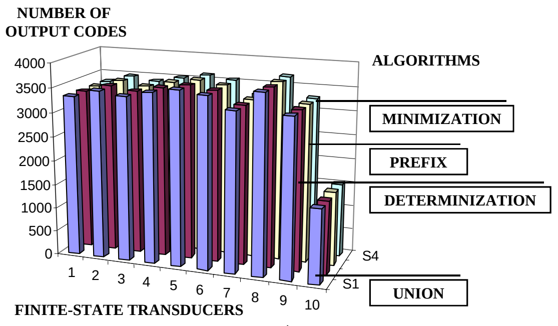


b)

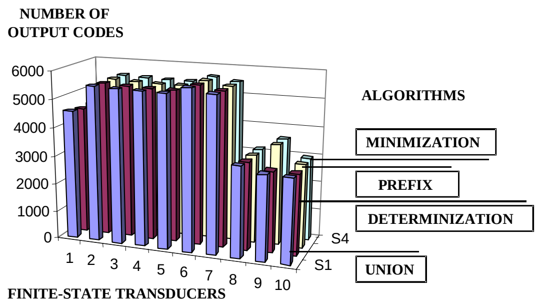
Figure 10: Achieved reduction of the number of states obtained in the second step of compilation process – 10 randomly chosen transducers (*a:* phonetic lexicon. *b:* morphology lexicon.)

	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 5: The f*inal finite-state transducers representing German phonetic (***FST_1***) and German morphology lexicon (***FST_2***).



a)



b)

Figure 11: Increasing number of output codes in the second step of compilation process – 10 randomly chosen transducers (*a:* phonetic lexicon. *b:* morphology lexicon.)

5 Conclusion

Performing *determinization* of finite-state transducer obviously results in significant decrease in number of states. One disadvantage of the determinization algorithm is that the outputs are pushed toward final states that create a long delay in emission. But using *prefix calculation* algorithm before classical minimization algorithms for automata, the problem can be efficiently resolved. An important characteristic of this algorithm is that it pushes back outputs as much as possible toward the initial state. As expected, the number of states remains unchanged after performing *prefix calculation* algorithm. The efficiency of this algorithm can be seen only after performing classical minimization algorithm, when much smaller number of states is obtained than in case if only determinization and minimization algorithms would be performed. From table 5 it can be seen that finite-state transducers can efficiently represent large lexicons. They provide fast look-up time, double side look-up, and compactness.

6 References

[1] Bruce William Watson, *Taxonomies and Toolkits of Regular Language Algorithms*, PhD Thesis, Eindhoven University of Technology and Computing Science, 1995.

[2] Watson, B.W., *A taxonomy of finite automata minimization algorithms*, Computing Science Report 93/44, Eindhoven University of Technology, The Nederlands, 1993.

[3] Mehryar Mohri, *On Some Applications of Finite-State Automata Theory to Natural Language Processing*, Natural Language Engineering 1, Cambridge University Press, 1996.

[4] Mehryar Mohri, *Minimization Algorithms for Sequential Transducers*, Theoretical Computer Science, 234:177-201, March 2000.

[5] Guenthner, F.&P. Maier, *Das CISLEX Woerterbuch system*, CIS-Bericht-94-76.

[6] Aho, Alfred V., John E. Hopcroft, and Jeffrey D. Ullman, *The design and analysis of computer algorithms*. Addison Wesley: Reading, MA 1974.

[7] Bauer, W, *On minimizing finite automata*, EATCS Bulletin, 35 1988.

[8] Berstel, Jean and Cristophe Reutenauer, *Rational Series and Their Languages*, Springer-Verlag: Berlin-New York 1988.

[9] Crochemore, Maxime, *Transducers and repetitions*, Theoretical Computer Science, 45 1986.

[10] Hopcroft, John E. and Jeffrey D. Ullman, *Intoduction to Automata Theory, Languages, and Computation*. Addison Wesley: Reading MA 1979.

[11] Kuich, Werner and Arto Salomaa, *Semirings, Automata, Languages*, Number 5 in EATCS Monographs on Theoretical Computer Science. Springer Verlag, Berlin, Germany 1986.

[12] Mehryar Mohri, *Language Processing with Weighted Transducers*, In Proceedings of the 8th annual Traitement Automatique des Langues Naturelles (TALN 2001). Tours, France, July 2001.