Representation of Large Lexica Using Finite-State Transducers for the Multilingual Text-to-Speech Synthesis Systems

Matej Rojc, Zdravko Kačič

Faculty of Electrical Engineering and Computer Science, University of Maribor
Smetanova 17, 2000 Maribor, Slovenia
matej.rojc@uni-mb.si,kacic@uni-mb.si

Abstract

Large external language resources used for multilingual text processing in TTS systems represent a big problem because of needed space and slow look-up time. Representation of large lexica using finite-state transducers is mainly motivated by considerations of space and time efficiency. In the paper we present a method and results of compiling large German phonetic and morphology lexica (CISLEX) [4] into corresponding finite-state transducers (FSTs), both with about 300.000 words. For both lexica a great reduction in size and optimal access time was achieved. The starting size for German phonetic lexicon was 12.526 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.

1. Motivation

Finite-state machines are 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 use of finite-state machines is very convenient since they allow one to describe easily most of the relevant local phenomena in the language. They provide also compact representation of language specific lexical rules needed for knowledge representation in the automatic multilingual 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.

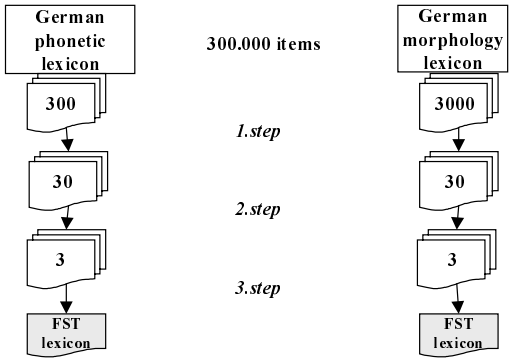

Figure 1: Lexica preparation.

In multilingual text-processing module for the multilingual TTS systems, external language resources (phonetic, morphology lexica etc.) represent a problem, regarding memory usage and time spend for lookup process. In the following we are presenting an approach for compiling such lexica into finite-state transducers that represent time and space optimal representation of such lexica. The effect of using finite-state transducers for representation of external language resources is great reduction of the memory usage required by the lexica and the optimal access time (required for obtaining information) that is independent from the size of the lexica. In the following sections the whole compilation process into finite-state transducers, and at the end also results obtained for the German lexica, will be presented.

2. Compilation of large scale lexica into finite-state transducers

2.1. Finite-state automata and finite-state transducers

When representing lexica 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. As newly developed lexica are more and more accurate, the number of codes can increase considerably. The increase in number of codes also increases the possible compacted size of such lexica. Also during the construction of the automaton one needs to distinguish different codes. The space required for an efficient hashing of the codes can also become costly. Available lexica that were used in this experiment suggest that the representation by automata would be less appropriate. Since morphological and phonetic lexica can be viewed as a list of pairs of strings, their representation using finite-state transducers seems to be very appropriate. The results at the end also confirm this assumption. Representation using finite-state transducers on the other hand also provides reverse look-up capability.

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

2.2. Compilation process

The methods used in the compilation of large scale lexica into finite-state transducers (FST) assume, that the lexica are given as large list of strings and not as a set of rules as considered

by Kaplan and Kay [1] for instance. Obviously morphological and phonetic lexica can be viewed as a list of pairs of strings and their representation using finite-state transducers seems to be very appropriate.

2.3. Lexica preparation

As with automata one cannot construct directly the sequential transducer representing a large-scale lexicon. The construction leads to a blow up for a large number of entries. So one needs first to split the lexicon into several parts, construct the corresponding sequential transducers, minimize them and then perform the union of these transducers and reminimize the resulting one (Fig.1).

2.4. Determinization of finite-state transducers

The used algorithm is close to the powerset construction used for determinizing automata [1]. 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. That's why only the longest common prefix of outputs can be kept and subsets are made of pairs (state, string). On Fig. 3 we can see the result of using the determinization algorithm on transducer in Fig. 2 (obtained using union operation). In this example the number of states of the determinized transducer T_2 is less than in T_1. The pseudocode for the algorithm to determinize a transducer T_1 is given below [1].

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**

$\sigma_2(q_2, a)$

10 $\leftarrow \bigwedge_{(q,a) \in J_1(a)} \left[w \cdot \bigwedge_{q' \in \delta_1(q,w)} \sigma_1(q,a,q') \right]$

$\delta_2(q_2, a)$

11 $\leftarrow \bigcup_{(q,w,q') \in J_2(a)} \{(q', [\sigma_2(q_2,a)]^{-1} w \cdot \sigma_1(q,a,q'))\}$

12 if ($\delta_2(q_2, a)$ is a new state)

13 then Enqueue(Q,$\delta_2(q_2, a)$)

14 Dequeue(Q)

At each step of the algorithm a new state q_2 is considered (line5). q_2 is a final state if it contains a pair (q,w) with q final in T_1. In that case, w is the final output at the state q_2. Then each input label a of the transitions leaving the states of the subset q_2 is considered (line 10). A transition is constructed from q_2 to $\delta_2(q_2,a)$ with output $\sigma_2(q_2,a)$. $\sigma_2(q_2,a)$ is 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. $\delta_2(q_2,a)$ is the subset made of pairs (q',w') where 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 output earlier in the algorithm. $[\sigma_2(q_2,a)]^{-1} w \sigma_1(q,a,q')$ is a well defined string since $[\sigma_2(q_2,a)]$ is a prefix of all $w\sigma_1(q,a,q')$ (line 10).

Experiments showed that this method is very efficient for constructing transducers representing large lexica. One disadvantage of this algorithm is that the outputs are pushed toward final states, which creates a long delay in emission. But sequential transducers can be minimized as we will show in the next sections. An important characteristic of that 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.

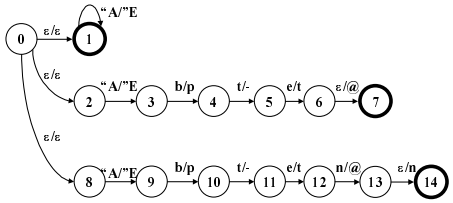

Figure 2: Union operation done on a few word items in the German phonetic lexicon (T_1).

2.5. Minimization of finite-state transducers

Sequential transducers allow very fast look-up. Minimization helps to make them also space efficient [2][3]. The minimization algorithm for sequential transducers involves the computation of the prefix of a non-deterministic automaton [2]. We present this algorithm first, as it is independent of the concept of sequential transducer. We then give a characterization of minimal sequential transducers and describe the entire algorithm allows to obtain these transducers.

In the following algorithm we denote by:

- G^T the transpose of G, namely 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:

$$\pi_u \leftarrow \left(\bigwedge_{\substack{t \in \mathrm{Trans}[u] \\ t.v \notin \mathrm{scc}}} t.l\, X_{t.v}\right) \wedge \left(\bigwedge_{\substack{t \in \mathrm{Trans}[u] \\ t.v \notin \mathrm{scc}}} t.l\right) \quad \text{if } u \notin F, \qquad (1)$$

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

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:

$$\forall t \in \mathrm{Trans}[u],\quad t.l \leftarrow \pi_u^{-1}\, t.l, \qquad \forall t \in \mathrm{Trans}^T[u],\quad t.l \leftarrow t.l\, \pi_u . \tag{2}$$

We can limit the number of times these two operations are performed by storing in an array N the number of empty labels leaving each state u of *scc*. As long as $N[u] \neq 0$, there is no use performing these operations as the value of π_u is ε. Also, if $N[u] = 0$ right after the computation of π_u, then π_u will remain equal to ε, as changes of variables will only affect suffixes of the transitions leaving u. We can store this information using an array F, in order to avoid performing *step 1* in such situations or when u is a final state. We used 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 additional array *INQ* indicating for each state u whether it is in Q.

We start the above operations by initializing N and F to 0 for all states in *scc*, and by enqueuing in Q an arbitrarily chosen state u of *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* 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 $Q = \varnothing$. This necessarily happens 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 have or $N[u] = 0$ 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. Thus, each state u is enqueued at most $(|P(u)|+2)$ times in Q, and after at most $(|Pmax|+2)$ steps we have $Q = 0$.

```
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
```

Once $Q = \varnothing$, it is easy to notice 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 thus 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*. This shows that the transformations described above do modify the transitions leaving or entering states of *scc* as desired. Thus, the above pseudocode gives an algorithm that computes $p(G)$ from G. In this algorithm, $V(G^{SCC})$ represents the set of states of the component graph of G, and, 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, gives $F[u]$ the value 1.

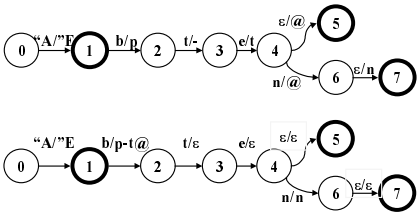

Figure 3: Finite-state transducers T_2 (above) and T_3 (below) obtained after performing determinization and prefixation algorithms on finite-state transducer showed in Fig. 2.

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

Given a *ST* T, the application of the *prefix_computation* algorithm [2] to the output automaton of T has no effect on the states of T or on its transition function. Only its output function σ 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 the minimization in the sense of automata to the resulting transducer. Fig. 3 below 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.

3. Results

For the lexica compilation we carried out, we used German large scale phonetic and morphology lexica (CISLEX). In compilation process we used a large set of programs written in C++, that can be used to perform efficiently many operations

on finite-state transducers and finite-state automata including determinization, minimization, union, intersection, compaction, prefixation, local extension and other algorithms.

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) [2][3]. Prefix computation algorithm was used before minimization algorithms and consists in pushing output labels towards the initial state as much as possible. The tests were performed on German phonetic and morphology lexica (CISLEX). All lexica 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 12.526 MB and 18.49 MB. Final size of corresponding finite-state transducer was 2.78 MB (120.386 states) for the first one and 6.33 MB (183.123) for the last one. Both lexica contained 300.000 items.

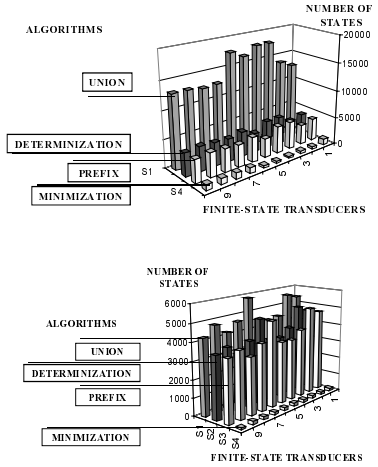

Figure 4: Achieved reduction of the number of states obtained in the first step of compilation process – 10 randomly chosen transducers (*Above:* phonetic lexicon. *Below:* morphology lexicon.)

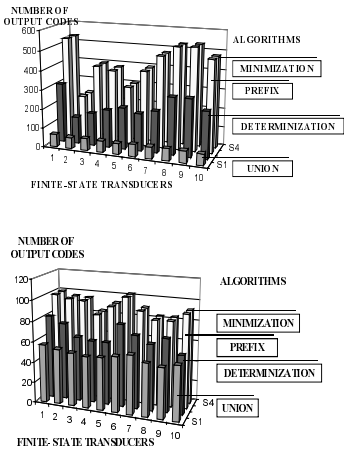

Figure 5: Increasing number of output codes in the first step of compilation process – 10 randomly chosen transducers (*Above:* phonetic lexicon. *Below:* morphology lexicon.)

The great reduction of the number of states is evident from Fig. 4. The number of states does not change after performing *prefix computation* algorithm. As can be seen on the next Fig.5, only the number of output codes is significantly increased. The effect of *prefix calculation* algorithm is 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.

In Table 5 the final results for the obtained finite-state transducers for german phonetic and morphology lexica are given.

Table 5: Final finite-state transducers representing German phonetic (***FST***$_1$) and German morphology lexicon (***FST***$_2$).

	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**

4. 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 creates a long delay in emission. But using *prefix calculation* algorithm before classical minimization algorithms for automata, the problem just mentioned is 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 lexica. They provide fast look-up time, double side look-up, and compactness.

5. References

[1] Mehryar Mohri, *On Some Applications of Finite-State Automata Theory to Natural Language Processing*, Natural Language Engineering 1, Cambridge University Press, 1995.

[2] Mehryar Mohri, *Minimization of Sequential Transducers*, Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching, pages 151-163. Springer Verlag, Berlin, 1994.

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

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