Reduction of Intermediate Alphabets in Finite-State Transducer Cascades

André Kempe

Xerox Research Centre Europe – Grenoble Laboratory
6 chemin de Maupertuis – 38240 Meylan – France
andre.kempe@xrce.xerox.com

Abstract

This article describes an algorithm for reducing the intermediate alphabets in cascades of finite-state transducers (FSTs). Although the method modifies the component FSTs, there is no change in the overall relation described by the whole cascade. No additional information or special algorithm, that could decelerate the processing of input, is required at runtime. Two examples from *Natural Language Processing* are used to illustrate the effect of the algorithm on the sizes of the FSTs and their alphabets. With some FSTs the number of arcs and symbols shrank considerably.

1. Introduction

This article describes an algorithm for reducing the intermediate alphabet occurring in the middle of a pair of finite-state transducers (FSTs) that operate in a cascade, i.e., where the first FST maps an input string to a number of intermediate strings and the second maps those to a number of output strings. With longer cascades, the algorithm can be applied pair-wise to all FSTs. Although the method modifies the component FSTs and component relations that they describe, there is no change in the overall relation described by the whole cascade. No additional information or special algorithm, that could decelerate the processing of input, is required at runtime.

Intermediate alphabet reduction can be beneficial for many practical applications that use FST cascades. In *Natural Language Processing*, FSTs are used for many basic steps (Karttunen *et al.*, 1996; Mohri, 1997), such as phonological (Kaplan & Kay, 1994) and morphological analysis (Koskenniemi, 1983), part-of speech disambiguation (Roche & Schabes, 1995; Kempe, 1997; Kempe, 1998), spelling correction (Oflazer, 1996), and shallow parsing (Koskenniemi *et al.*, 1992; Aït-Mokhtar & Chanod, 1997). Some of these applications, such as shallow parsing, use FST cascades. Others could jointly be used in a cascade. In these cases, the proposed method can reduce the sizes of the FSTs.

The described algorithm has been implemented.

1.1. Conventions

The conventions below are followed in this article.

Examples and figures: Every example is shown in one or more figures. The first figure usually shows the original network or cascade. Possible following figures show modified forms of the same example. For example, Example 1 is shown in Figure 1 and Figure 2.

Finite-state graphs: Every network has one initial state, labeled with number 0, and one or more final states marked by double circles. The initial state can also be final. All other state numbers and all arc numbers have no meaning for the network but are just used to reference a state or an arc from within the text. An arc with *n* labels designates a set of *n* arcs with one label each that all have the same source and destination. In a symbol pair occurring as an arc label, the first symbol is the input and the second the output symbol. For example, in the symbol pair `a:b`, `a` is the input and `b` the output symbol. Simple (i.e. unpaired) symbols occurring as an arc label, represent identity pairs. For example, `a` means `a:a`. The question mark, "?", denotes (and matches) all unknown symbols, i.e., all symbols outside the alphabet of the network.

Input and output side: Although FSTs are inherently bidirectional, they are often intended to be used in a given direction. The proposed algorithm is performed wrt. the direction of application. In this article, the two sides (or tapes or levels) of an FST are referred to as *input side* and *output side*.

2. Previous Work

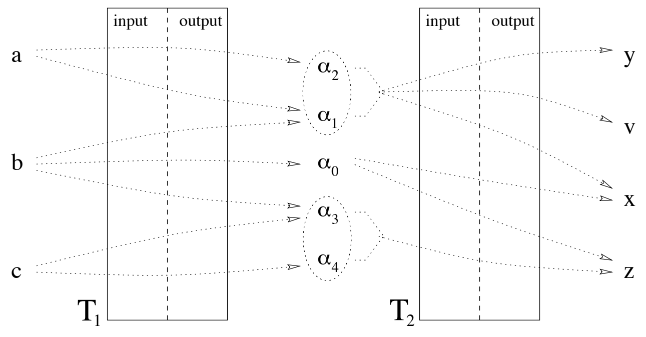
The below described algorithm of intermediate alphabet reduction is related to the idea of *label set reduction* (Koskenniemi, 1983; Karttunen *et al.*, 1987). The later is applied to a single FST or automaton. It groups all arc labels into equivalence classes, regardless whether these are atomic labels (e.g. "`a`"), identity pairs (e.g. "`a:a`"), or non-identity pairs (e.g. "`a:x`"). Labels that always co-occur on arcs with the same source and destination state, are put into the same equivalence class. One label is then selected from every class to represent the class. All other labels are removed from the alphabet, and the corresponding arcs are removed from the network, which can lead to a considerable size reduction. Label set reduction is reversible, based on the information about the equivalence classes. At runtime, this information is required together with a special algorithm to interpret every label in the network as the set of labels in the corresponding equivalence class. For example, if the label `a` represents the class {`a`, `a:b`, `b`, `c:z`} then it must map `c`, occurring at the input, to `z`.

3. Reduction of Intermediate Alphabets

The algorithm of intermediate alphabet reduction is applied to a pair of FSTs that operate in a cascade rather than to a single FST. It reduces the intermediate alphabet between the two FSTs without necessarily reducing the label sets of the FSTs. With longer cascades, the algorithm can be applied pair-wise to all FSTs. Although the component FSTs and component relations that they describe are (irreversibly) modified, there is no change in the overall relation described by the whole cascade. No additional information or special algorithm, that could decrease the processing speed, is required at runtime. The fact that every intermediate symbol actually represents a set of (one or more) symbols, can be neglected at that point. Every symbol will be considered at runtime just as itself.

3.1. Alphabet Reduction in Transducer Pairs

We will first describe the algorithm for an FST pair where the two FSTs, T_1 and T_2, operate in a cascade, i.e., T_1 maps an input string to a number of intermediate strings which, in turn, are mapped by T_2 to a number of output strings.



Equivalence classes: $\{\alpha_0\}, \{\alpha_1, \alpha_2\}, \{\alpha_3, \alpha_4\}$

Figure 1: Transducer pair (Example 1)

Example 1 shows an FST pair and part of its input, intermediate, and output alphabet (Fig. 1). Suppose, both intermediate symbols α_1 and α_2 are always mapped to the same output symbol which can be `y`, `v`, or `x` depending on the context. This means, α_1 and α_2 constitute an equivalence class. There may be another class formed by α_3 and α_4. If we are not interested in the actual intermediate symbols but only in the final output, we can select one member symbol of every class to represent the class, and replace all other symbols by the representative of their class. In Example 1, this means that α_2 is replaced by α_1, and α_4 by α_3 (Fig. 2).

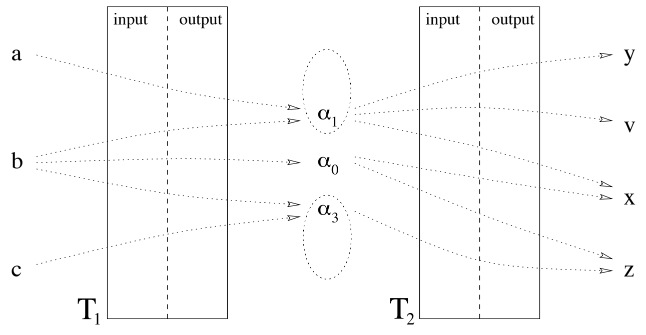


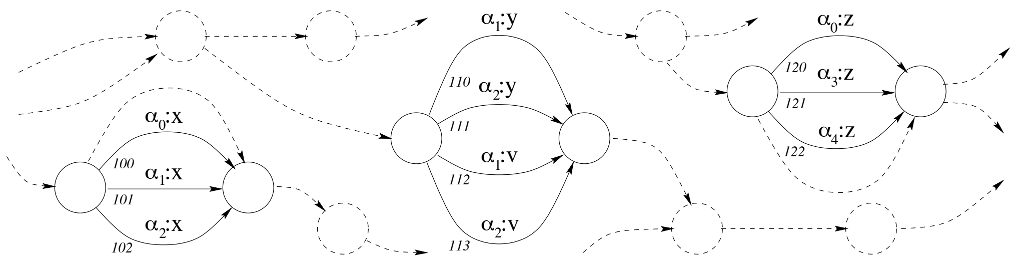
Figure 2: Transducer pair with reduced intermediate alphabet (Example 1)

The algorithm works as follows. First, equivalence classes are constituted among the input symbols of T_2 (Fig. 3).[1] For this purpose, all symbols are put initially into one single class which is then more and more partitioned as the arcs of T_2 are inspected. The construction of equivalence classes terminates either when all arcs have been inspected or when the maximal partitioning (into singleton classes) is reached.

Two symbols, α_i and α_j, are considered equivalent if for every arc with α_i as input symbol, there is another arc with α_j as input symbol and vice versa, such that both arcs have the same

[1]In Figure 3, 4, and 7 only transitions and states that are relevant for the current purpose are represented by solid arcs and circles, and all the others by dashed arcs and circles.

source and destination state and the same output symbol. In Example 2, we constitute the equivalence classes $\{\alpha_0\}$, $\{\alpha_1, \alpha_2\}$, and $\{\alpha_3, \alpha_4\}$ (Fig. 3). Here, α_0 constitutes a class on its own because it first co-occurs with α_1 and α_2 in the arc set {*100, 101, 102*}, and later with α_3 and α_4 in the arc set {*120, 121, 122*}.



Equivalence classes: $\{\underline{\alpha_0}\}, \{\underline{\alpha_1}, \alpha_2\}, \{\underline{\alpha_3}, \alpha_4\}$

Figure 3: Second transducer of a pair (Example 2)

Subsequently, all occurrences of intermediate symbols are replaced by the representative of their class. In Example 2, we selected the first member of each class as its representative. The replacement must be performed both on the output side of T_1 and on the input side of T_2. Figure 4 shows the effect of this replacement on T_2 in Example 2.

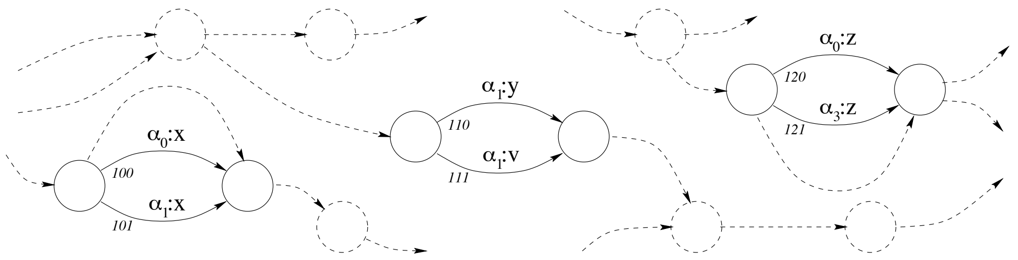


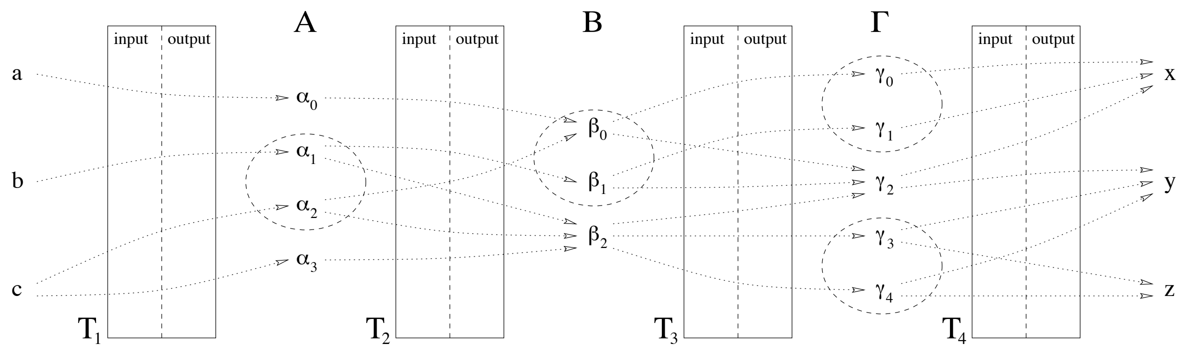
Figure 4: Second transducer of a pair with reduced intermediate alphabet (Example 2)

The algorithm can be applied to any pair of FSTs: They can be ambiguous or they can contain "?", the unknown symbol, or ϵ, the empty string, or even ϵ-loops. Although ϵ and ? cannot be merged with other symbols, this does not represent a restriction for the type of the FSTs.

3.2. Alphabet Reduction in Transducer Cascades

In a cascade, intermediate alphabet reduction can be applied pair-wise to all FSTs starting from the end of the cascade. Example 3 shows a cascade of four FSTs with the intermediate alphabets A, B, and Γ (Fig. 5). The reduction is first applied to the last intermediate alphabet, Γ, between T_3 and T_4. Suppose, there are three equivalence classes in Γ, namely $\{\gamma_0, \gamma_1\}$, $\{\gamma_2\}$, and $\{\gamma_3, \gamma_4\}$. According to the above method, the class $\{\gamma_0, \gamma_1\}$ can be represented by γ_0, $\{\gamma_2\}$ by γ_2, and $\{\gamma_3, \gamma_4\}$ by γ_3. This means, all occurrences of γ_1 are replaced by γ_0 and all occurrences of γ_4 by γ_3, both on the output side of T_3 and on the input side of T_4 (Fig. 5, 6). Consequently, β_1 in the preceding alphabet, B, will now be mapped to γ_0 instead of γ_1, and β_2 to γ_3 instead of γ_4. The latter mapping actually exists already.

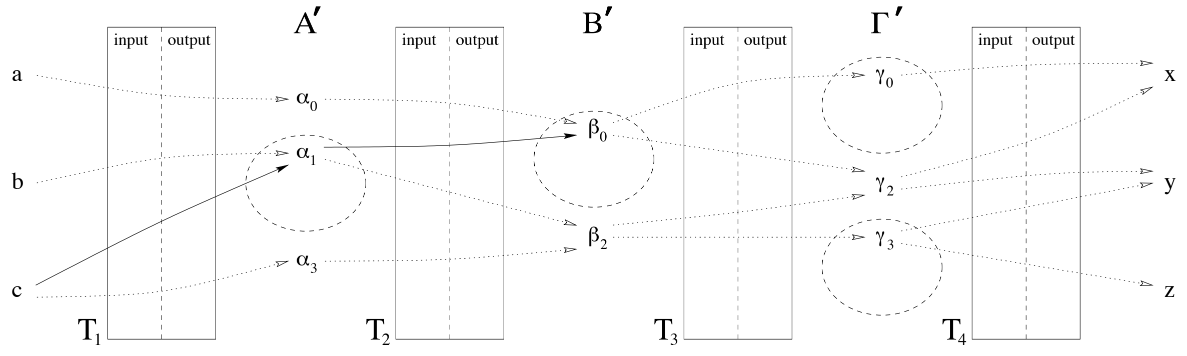
Subsequently, the preceding intermediate alphabet, B, is reduced. It may contain two equivalence classes, $\{\beta_0, \beta_1\}$ and $\{\beta_2\}$. Both members of $\{\beta_0, \beta_1\}$ are at present mapped either to



Equivalence classes in different alphabets:
A:$\{\{\alpha_0\},\{\alpha_1,\alpha_2\},\{\alpha_3\}\}$, B:$\{\{\beta_0,\beta_1\},\{\beta_2\}\}$, Γ:$\{\{\gamma_0,\gamma_1\},\{\gamma_2\},\{\gamma_3,\gamma_4\}\}$

Figure 5: Transducer cascade (Example 3)

γ_0 (previously $\{\gamma_0,\gamma_1\}$) or to γ_2, depending on the context. Note, the class $\{\beta_0,\beta_1\}$ can only be constituted if the alphabet Γ has been previously reduced and $\{\gamma_0,\gamma_1\}$ was replaced by γ_0. Finally, we reduce the intermediate alphabet A, based on its equivalence classes $\{\alpha_0\}$, $\{\alpha_1,\alpha_2\}$, and $\{\alpha_3\}$ (Fig. 6).



Solid lines mark modified relations.

Figure 6: Transducer cascade with reduced intermediate alphabets (Example 3)

The best overall reduction of the cascade is guarantied if the algorithm starts with the last intermediate alphabet, and processes all alphabets in reverse order. This is for the following reason: Suppose, an FST inside a cascade maps α_0 to β_0 and α_1 to β_1 on two arcs with the same source and destination state (Fig. 7). If the alphabet B is processed first and β_0 and β_1 are reduced to one symbol (e.g. β_0) then α_0 and α_1 will have equal output symbols and may therefore become reducible to one symbol (e.g. α_0), depending on their other output symbols in this FST. If β_0 and β_1 cannot be reduced then subsequently α_0 and α_1 cannot be either because they have different output symbols, at least in this case. This means, reducing B first is either beneficial or neutral for the subsequent reduction of A. Not reducing B first is always neutral for A. Therefore, the intermediate alphabets of a cascade should be reduced all in reverse order.

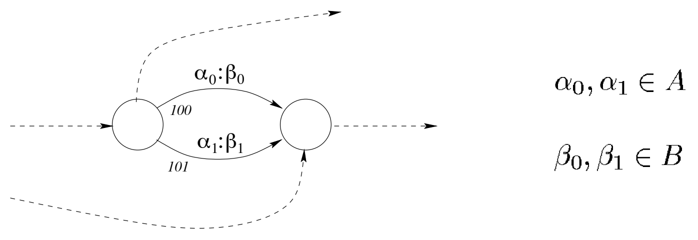


Figure 7: Transducer inside a cascade (Example 4)

4. Complexity

The running time complexity of constructing equivalence classes is in the general case quadratic to the number of outgoing arcs on every state of the second FST, T_2, of a pair:

$$O(\sum_{q \in Q_2} |E(q)|^2) \tag{1}$$

where $|E(q)|$ is the number of outgoing arcs of state q. This is because in general every arc of q must be compared to every other arc of q. In the extremely unlikely worst case when all arcs have the same source state, this complexity is quadratic to the total number of arcs in T_2 :

$$O(|E_2|^2) \tag{2}$$

The running time complexity of replacing symbols by the representatives of their equivalence class is in any case linear to the total number of arcs in T_1 and T_2 :

$$O(|E_1| + |E_2|) \tag{3}$$

The space complexity of constructing equivalence classes is linear to the size of the intermediate alphabet that is to be reduced:

$$O(|\Sigma_{\text{mid}}|) \tag{4}$$

The replacement of symbols by the representatives of their class requires no additional memory space.

5. Evaluation

Table 1 shows the effect of intermediate alphabet reduction on the sizes of FSTs and their alphabets in a cascade. The twelve FSTs are used for shallow parsing of French text, e.g., for the marking of noun phrases, clauses, and syntactic relations such as subject, inverted subject, or object (Aït-Mokhtar & Chanod, 1997). An input string to the cascade consists of surface word forms, lemmas, and part-of-speech tags of a whole sentence. The output is ambiguous and consists alternatively of a parse (only one per sentence) or of a syntactic relation such as "`SUBJ(Jean,mange)`" (several per sentence).

With some FSTs the reduction was considerable. For example, the number of arcs of the largest FST (#10) was reduced from 1 156 053 to 498 506. In some other cases no reduction was possible. For the whole cascade the number of arcs has shrunk from 2 704 047 to 1 731 831 which represents a reduction of 36%.

FST	originally #states	originally #arcs	originally #symbols input	originally #symbols output	after reduction #states	after reduction #arcs	after reduction #symbols input	after reduction #symbols output
1	10 487	404 903	138	137	10 487	404 903	138	137
2	604	28 569	136	135	604	28 569	136	135
3	27 704	225 215	11	10	27 704	225 215	11	10
4	3 613	61 259	23	23	3 613	61 259	23	23
5	1 276	128 222	146	142	1 276	**124 754**	**143**	142
6	3 293	29 079	12	12	3 293	29 079	12	12
7	5 544	166 704	34	33	5 544	**90 024**	**19**	**18**
8	396	19 008	48	48	396	**12 276**	**31**	**31**
9	7 009	370 419	54	54	7 009	**204 411**	**30**	**27**
10	6 033	1 156 053	158	158	6 033	**498 506**	**66**	**65**
11	573	114 328	171	171	573	**52 801**	**78**	**45**
12	2	288	144	16	2	**34**	**17**	16
total	66 534	2 704 047	—	—	66 534	**1 731 831**	—	—

Bold numbers denote modifications.

Table 1: Sizes of transducers and of their alphabets in a cascade

Each of these FSTs has its own alphabet which can contain "?" that matches any symbol that is not explicitly mentioned in the alphabet of the FST. Therefore, an FST can have a different number of output symbols than the following FST has input symbols because a given symbol may be part of the unknow alphabet in one FST but not in the other.

Lexicon	FST	originally #states	originally #arcs	originally #symbols input	originally #symbols output	after reduction #states	after reduction #arcs	after reduction #symbols input	after reduction #symbols output
E	T	62 120	156 757	224	298	—	—	—	—
n	T_1	75 900	191 687	224	8 874	**73 780**	**181 829**	224	**3 042**
g	T_2	16 748	36 737	8 729	153	16 748	**24 483**	**2 897**	153
	T_1+T_2	92 648	228 424	—	—	**90 528**	**206 312**	—	—
F	T	55 725	130 139	224	269	—	—	—	—
r	T_1	61 467	164 819	224	11 420	**59 697**	**159 187**	224	**2 680**
e	T_2	6 814	59 587	11 241	90	6 814	**42 007**	**2 501**	90
	T_1+T_2	68 281	224 406	—	—	**66 511**	**201 194**	—	—

T *.... original FST,* T_1, T_2, T_1+T_2 *.... first and second FST from factorization and sum of both. Bold numbers denote modifications.*

Table 2: Sizes of transducers and of their alphabets in factorization

Intermediate alphabet reduction can also be useful with the factorization of a finitely ambiguous FST into two FSTs, T_1 and T_2, that operate in a cascade (Kempe, 2000). Here, T_1 maps any input symbol that originally has ambiguous output, to a unique intermediate symbol which is then mapped by T_2 to a number of different output symbols. T_1 is unambiguous. T_2 retains the ambiguity of the original FST, but it is *fail-safe* wrt. T_1. This means, the application of T_2

to the output of T_1 never leads to a state that does not provide a transition for the next input symbol, and always terminates in a final state. This factorization can create many redundant intermediate symbols, but their number can be reduced by the above algorithm. Table 2 shows the effect of alphabet reduction on FSTs resulting from the factorization of an English and a French lexicon.

Since T_1 is unambiguous it can be further factorized into a left- and a right-sequential FST that jointly represent a bimachine (Schützenberger, 1961). The intermediate alphabet of a bimachine, however, can be limited to the necessary minimum already during factorization (Elgot & Mezei, 1965; Berstel, 1979; Reutenauer & Schützenberger, 1991; Roche, 1997), so that the above algorithm is of no use in this case.

6. Conclusion

The article described an algorithm for reducing the intermediate alphabets in an FST cascade. The method modifies the component FSTs but not the overall relation described by the whole cascade. The actual benefit consists in a reduction of the sizes of the FSTs.

Two examples from Natural Language Processing have been used to illustrate the effect of the alphabet reduction on the sizes of FSTs and their alphabets, namely a cascade for shallow parsing of French text and FST pairs resulting from the factorization of lexica. With some FSTs the number of arcs and symbols shrank considerably.

Acknowledgments

I wish to thank Andreas Eisele (XRCE Grenoble) and the anonymous reviewers of my article for their valuable comments and advice.

References

AÏT-MOKHTAR S. & CHANOD J.-P. (1997). Incremental finite-state parsing. In *Proceedings of the 5th International Conference on Applied Natural Language Processing (ANLP)*, p. 72–79, Washington, DC, USA: ACL.

BERSTEL J. (1979). *Transductions and Context-Free Languages*. Number 38 in Leitfäden der angewandten Mathematik und Mechanik (LAMM). Studienbücher Informatik. Stuttgart, Germany: Teubner.

ELGOT C. C. & MEZEI J. E. (1965). On relations defined by generalized finite automata. *IBM Journal of Research and Development*, p. 47–68.

KAPLAN R. M. & KAY M. (1994). Regular models of phonological rule systems. *Computational Linguistics*, **20**(3), 331–378.

KARTTUNEN L., CHANOD J.-P., GREFENSTETTE G. & SCHILLER A. (1996). Regular expressions for language engineering. *Natural Language Engineering*, **2**(4), 305–328.

KARTTUNEN L., KOSKENNIEMI K. & KAPLAN R. M. (1987). A compiler for two-level phonological rules. In DALRYMPLE M. ET AL., Ed., *Tools for Morphological Analysis*, volume CSLI-86-59 of *CSLI Reports*. Xerox Palo Alto Research Center and Center for the Study of Language and Information, Stanford University.

KEMPE A. (1997). Finite-state transducers approximating hidden markov models. In *Proceedings of the 35th Annual Meeting*, p. 460–467, Madrid, Spain: ACL. Also in: `cmp-lg/9707006`.

KEMPE A. (1998). Look-back and look-ahead in the conversion of hidden markov models into finite-state transducers. In *Proceedings of the 3rd International Conference on New Methods in Natural Language Processing (NeMLaP)*, p. 29–37, Sydney, Australia: ACL. Also in: `cmp-lg/9802001`.

KEMPE A. (2000). Factorization of ambiguous finite-state transducers. In *Pre-Proceedings of the 5th International Conference on Implementation and Application of Automata (CIAA)*, p. 157–164, London, Ontario, Canada: The University of Western Ontario. Final Proceeding to appear in: *Springer-Verlag Lecture Notes in Computer Science*.

KOSKENNIEMI K. (1983). *Two-level Morphology: A General Computational Model for Word-Form Recognition and Production*. Publications 11, University of Helsinki, Department of General Linguistics, Helsinki, Finland.

KOSKENNIEMI K., TAPANAINEN P. & VOUTILAINEN A. (1992). Compiling and using finite-state syntactic rules. In *Proceedings of the 15th International Conference on Computational Linguistics (CoLing)*, volume 1, p. 156–162, Nantes, France: ACL.

MOHRI M. (1997). Finite-state transducers in language and speech processing. *Computational Linguistics*, **23**(2), 269–312.

OFLAZER K. (1996). Error-tolerant finite state recognition with applications to morphological analysis and spelling correction. *Computational Linguistics*, **22**(1), 73–89.

REUTENAUER C. & SCHÜTZENBERGER M. P. (1991). Minimization of rational word functions. *SIAM Journal of Computing*, **20**(4), 669–685.

ROCHE E. (1997). Compact factorization of finite-state transducers and finite-state automata. *Nordic Journal of Computing*, **4**(2), 187–216.

ROCHE E. & SCHABES Y. (1995). Deterministic part-of-speech tagging with finite-state transducers. *Computational Linguistics*, **21**(2), 227–253.

SCHÜTZENBERGER M. P. (1961). A remark on finite transducers. *Information and Control*, **4**, 185–187.