A GENERALIZED CONSTRUCTION OF INTEGRATED SPEECH RECOGNITION TRANSDUCERS

Cyril Allauzen *Mehryar Mohri* *Michael Riley* * *Brian Roark*

AT&T Labs – Research
180 Park Avenue
Florham Park, NJ 07932-0971, USA
{allauzen, mohri, roark}@research.att.com

ABSTRACT

We showed in previous work that *weighted finite-state transducers* provide a common representation for many components of a speech recognition system and described general algorithms for combining these representations to build a single optimized and compact transducer integrating all these components, directly mapping from HMM states to words. This approach works well for certain well-controlled input transducers, but presents some problems related to the efficiency of *composition* and the applicability of *determinization* and *weight-pushing* with more general transducers. We generalize our prior construction of the integrated speech recognition transducer to work with an arbitrary number of component transducers and, to a large extent, release the constraints imposed to the type of input transducers by providing more general solutions to these problems. This generalization allowed us to deal with cases where our prior optimization did not apply. Our experiments in the AT&T HMIHY 0300 task and an AT&T VoiceTone task show the efficiency of our generalized optimization technique. We report a 1.6 recognition speed-up in the HMIHY 0300 task, 1.8 speed-up in a VoiceTone task using a word-based language model, and 1.7 using a class-based model.

1. MOTIVATION

In previous work, we showed that *weighted finite-state transducers* provide a common and natural representation for many components of a speech recognition system, e.g., HMMs, context-dependency, pronunciation dictionaries, and language models [8]. We also described general algorithms for combining these representations flexibly and efficiently and showed that they can be used to build a single optimized transducer that integrates these components, directly mapping from HMM states to words [9, 10]. In this method, weighted transducer *composition* is used to combine the component transducers, while *determinization*, *minimization* and *weight-pushing* optimize the result in time and space. The resultant transducer has a standardized representation, unique up to state renumbering.

This approach works well for certain well-controlled input transducers, but presents some problems with more general transducers. These problems are related to composition, determinization, and weight-pushing.

Composition can use unacceptable amounts of time and space when there are significant delays in matching due to ϵ-transitions. In simple cases, this can be avoided by a careful construction of the component transducers. In practice, inexperienced users are often not fully aware of this. For example, they may place the output labels in the lexicon transducer L at the ends of words rather than at the beginning, which can significantly slow down composition. For more complex transducers, trying to manually place the input and output labels for the best effect becomes difficult.

Not all transducers are determinizable. The lexicon transducer, for example, is not determinizable if it contains homophones. In our prior construction, we added disambiguation symbols at word ends as needed to solve this problem. For more general transducer inputs, this is insufficient. A related problem is that the prior construction assumed that only L was non-determinizable in our composition chain while we might wish to use other information sources represented by non-determinizable transducers.

Not all transducers can be pushed over the (log) probability semiring, in particular because not all transducers can have their weights redistributed along their paths so that they form a stochastic distribution. A trivial example is a self-loop with weight greater than one. In practice, there are several common causes of non-stochastic knowledge sources: unweighted word grammars, e.g., hand-crafted grammars; silence models with free cost loops; unweighted or unnormalized multiple pronunciations; compact approximate automata representation of n-gram models; and *ad hoc* word or phone insertion penalties.

One way to deal with these problems would be to allow only very limited kinds and number of knowledge sources and precisely stipulate how each is constructed as a transducer prior to their combination and optimization. But much of the inherent generality of weighted transducer algorithms would be lost by such restrictions. Instead, we consider here how to generalize our constructions so that as wide a range of component transducers as possible can be used successfully as inputs. This generalization will permit the more efficient use of current models and of hopefully yet more innovative future models.

We present our generalized construction technique in detail in section 4 and show how,within the general framework considered, it provides solutions to each of the problems just mentioned. This culminates in the description of `dmake`, a utility from the AT&T Decoder Library (DCD Library), that takes a very general set of transducer inputs and constructs an optimized recognition transducer from them (Section 5). We also present in that Section the results of experiments in several spoken-dialog applications tasks that show very substantial recognition speed-ups compared to the previous optimizations (that were limited due to the kinds of issues mentioned above). We also demonstrate the efficiency of the construction of recognition transducers from component transducers.

We begin with some preliminary definitions of weighted transducers needed in the remainder of the paper (Section 2) and a brief description of some basic algorithms needed in our generalized construction (Section 3) that are also of independent interest.

2. PRELIMINARIES

Weighted automata are automata in which the transitions carry in addition to the usual alphabet symbols some weights elements of a *semiring* [5]. A semiring is a ring that may lack negation. It has two associative operations $\oplus$ and $\otimes$ with identity elements $\overline{0}$ and $\overline{1}$. $\otimes$ distributes over $\oplus$ and $\overline{0}$ is an annihilator. The weights used in speech recognition often represent probabilities. The appropriate semiring to use is then the *probability semiring* $(\mathbb{R}, +, \cdot, 0, 1)$. For numerical stability, implementations often replace probabilities with log probabilities. The appropriate semiring to use is then the *log semiring*: $(\mathbb{R} \cup \{\infty\}, \oplus_l, +, \infty, 0)$, with: $\forall a, b \in \mathbb{R} \cup \{\infty\}, a \oplus_l b = -\log(exp(-a) + exp(-b))$, where by convention $exp(-\infty) = 0$, and $-\log(0) = \infty$. The log semiring is the image by $\log$ of the semiring $(\mathbb{R}, +, \cdot, 0, 1)$. When log probabilities are used and a Viterbi approximation is assumed, $\oplus_l$ is replaced by $\min$ and the appropriate semiring is the *tropical*

*This author's new address is: Google, Inc, 440 Broadway, New York, NY 10018, riley@google.com.

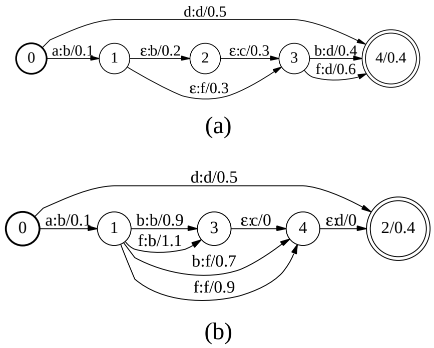

Fig. 1. Input ϵ-normalization in the tropical semiring. (a) Weighted transducer T_1. (b) Weighted transducer T_2 equivalent to T_1 output of the ϵ-normalization algorithm.

semiring $(\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$.

A *weighted transducer* $T = (\Sigma, \Delta, Q, I, F, E, \lambda, \rho)$ over $\mathbb{K}$ is a 7-tuple where Σ is a finite input alphabet, Δ is a finite output alphabet, Q is a finite set of states, $I \subseteq Q$ the set of initial states, $F \subseteq Q$ the set of final states, $E \subseteq Q \times \Sigma \times \Delta \times \mathbb{K} \times Q$ a finite set of transitions, $\lambda : I \rightarrow \mathbb{K}$ the initial weight function mapping I to $\mathbb{K}$, and $\rho : F \rightarrow \mathbb{K}$ the final weight function mapping F to $\mathbb{K}$ [12, 5]. *Weighted automata* can be defined in a similar way by simply omitting the output labels. A *path* $\pi = e_1 \cdots e_k$ in A is an element of E^* with consecutive transitions. A *successful path* is a path from an initial state to a final state.

3. BASIC ALGORITHMS

This section briefly describes several basic algorithms needed for our generalized construction of an integrated recognition transducer: *input ϵ-normalization* and *synchronization* are used in our optimization to normalize the positions of input and output labels of acyclic transducers, and a *symbol insertion* based on weighted transducer *determinization* used to ensure the determinizability of the closure of an acyclic transducer.

3.1. Input ϵ-Normalization

Input ϵ-normalization [6] is an algorithm used in our optimizations of the acyclic component transducers. It consists of normalizing the input transducer T_1 in the following way. The output transducer T_2 is equivalent to T_1, it has no ϵ-transitions, and along any of its successful paths, no transition with input label different from ϵ is preceded with a transition with input label ϵ. Figures 1(a)-(b) illustrate the application of this algorithm to the transducer of Figure 1(a).

3.2. Synchronization

There exists a general algorithm for the *synchronization of weighted transducers* [7]. The objective of the algorithm is to synchronize, to the extent that it is possible, the consumption of non-ϵ symbols by the input and output tapes of a weighted transducer. Figures 2(a)-(b) illustrate the algorithm and its application to the transducer of Figure 2(a). Each state of the resulting transducer T' corresponds to a triplet (q, x, y) where q is a state of the original machine T and where x and y are residual input and output strings used to create synchronized outgoing transitions.

3.3. Determinization with Symbol Insertion

Determinization can be applied to any acyclic transducer T to create an equivalent transducer T' that is *finitely subsequential*, i.e., a transducer with deterministic input and a finite number of string outputs at each final state. The algorithm can be augmented to insert a new and distinct input symbol corresponding to each of the final output strings at final states. The alphabet of the resulting transducer T'' has a finite number of additional auxiliary symbols.

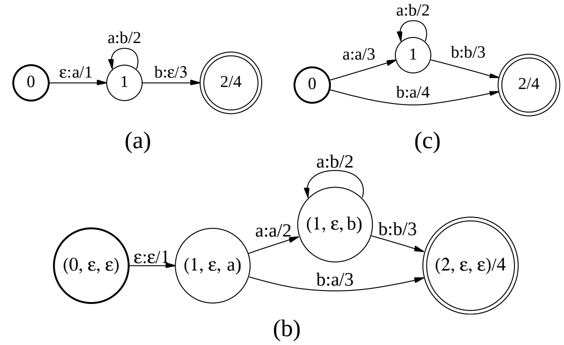

Fig. 2. (a) Weighted transducer T_1 over the tropical semiring. (b) Equivalent synchronized weighted transducer T_2. (c) Synchronized weighted transducer T_3 equivalent to T_1 and T_2 obtained by ϵ-removal from T_2.

T'' is unambiguous and the input label of each of its successful paths ends with an auxiliary symbol. It is not hard to show that the closure of T'' is determinizable. Figures 3(c)-(d) illustrate the application of the algorithm to the transducer of Figure 3(c).

4. GENERALIZED CONSTRUCTION

Our generalized construction applies to an arbitrary number n of transducers T_k, $k = 1, \ldots, n$, representing the information sources used by the system in the order of the numbering of the transducers. The essential scheme is similar to our prior construction [8]. Composition is applied right-to-left to combine information sources, with each composition step being followed by determinization to reduce redundancy. The resulting optimized integrated transducer N is thus constructed in the following way:

$$N = \pi_\epsilon(det(\tilde{T}_n \circ det(\tilde{T}_{n-1} \circ \cdots det(\tilde{T}_2 \circ \tilde{T}_1) \cdots)))$$

where the tilde operator indicates that disambiguation symbols are inserted so the transducer admits determinization or passes through the disambiguation symbols of prior stages, det stands for the weighted determinization algorithm applied in the log semiring and π_ϵ replaces the disambiguation symbols with ϵ transitions. Additionally, weighted minimization can be used to reduce the size of all (encoded) intermediate and final transducers.

Before applying the general construction just outlined, the component transducers T_k must be optimized. Ideally, arbitrary input transducers T_k could be considered. In fact, it helps to exploit the prior knowledge about these transducers to apply the appropriate optimizations in the most efficient way. As such, we distinguish three types of component transducers: *acyclic*, *bi-determinizable*, and *arbitrary* and describe in detail the optimization algorithms applied to each type.

No optimization algorithms are applied to *arbitrary* component transducers. We assume that the user ensured that these transducers are build adequately for composition, and that the resulting transducer after composition is determinizable. *Bi-determinizable* component transducers are simply determinized on their output side. No specific assumption is made about an *acyclic* component transducer, e.g., the relative position of the input and output labels, the weights, or topology of the machine. To deal with such arbitrary transducers, we use several algorithms to optimize and suitably disambiguate them, in particular the basic algorithms presented in the previous section. The exact sequence of algorithms applied are: input ϵ-normalization, determinization with symbol insertion — to ensure determinizability after closure, synchronization — to optimize the positions of the output labels for composition, ϵ-removal, then determinization, minimization and weight pushing applied to the transducer encoded as an acceptor (each pair of input and output labels is treated as one symbol), closure, and reverse ϵ-removal — to create a compact representation of the transducer. This construction is illustrated in Figure 3. The next sections describe our solutions to the problems previously mentioned.

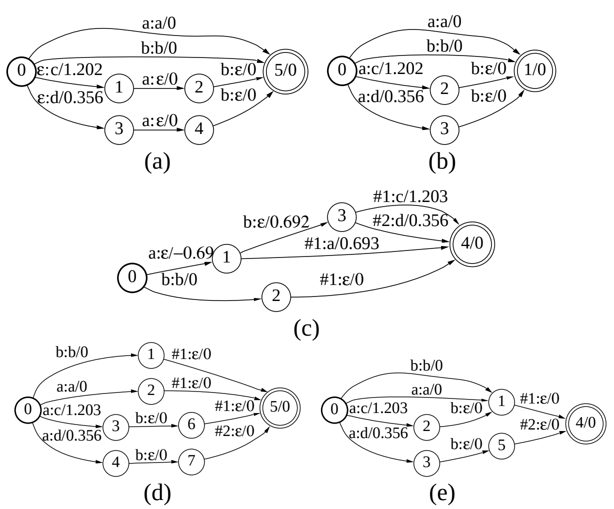

Fig. 3. (a) Acyclic weighted transducer T_1 over the log semiring. (b) Weighted transducer T_2 obtained from T_1 after input ϵ-normalization. (c) Weighted transducer T_3 obtained from T_2 by determinization with symbol insertion. (d) Weighted transducer T_4 obtained from T_3 by synchronization and ϵ-removal. (e) Weighted transducer T_5 obtained by applying to the transducer T_4 encoded as an acceptor (each pair of input and output labels is treated as one symbol) determinization, minimization and weight pushing.

4.1. Solutions to Composition- and Determinization-Related Problems

Each component transducer in the composition chain is determinizable: bi-determinizable components are determinizable by definition and the processing of acyclic transducers presented in the previous section ensure that their closure is determinizable. Hence, we need to ensure (1) that the results of the intermediary compositions are indeed determinizable, and (2) that the new input symbols that have been inserted in the acyclic component transducers do no disrupt the composition chain. To solve both issues, each transducer in the composition chain is modified to map each symbol out of its output alphabet to a new input symbol. This can be done by simulating at every state loops mapping out-of-alphabet output symbols to new input symbols. Note that we have also presented elsewhere a general technique for dealing with non-determinizable transducers [1].

4.2. Solutions to Weight-Pushing-Related Problems

4.2.1. Silence modeling

Silence tokens are typically not modeled within language models. They are added after the fact, most commonly as free cost loops at various states in the automata representation of the model, such as the initial, final, and unigram states. This makes the model non-stochastic.

One solution consists of force-aligning the training transcripts to the training audio, which produces silence tokens in the transcript at the points that best align the transcript to the audio. The resulting annotation can be used to explicitly include such tokens in the language model. With that approach, silences are treated just like other words, their probabilities are conditioned by previous words and they condition the probabilities of the words that follow. Hand-crafted grammars, multiple pronunciations, and hand-crafted silence models all admit to this solution. However, there are several problems with this approach. First, it requires a forced alignment to be run for every model that is built, which may or may not be possible. More importantly, silences are not distributed like words, and treating them as such can disrupt valid lexical dependencies by placing silence tokens, which now participate in the Markov chain, between words.

Non-stochasticity arises because words can be variously realized, i.e. each word can be followed by zero or more silence tokens. To make the model stochastic, the probability mass associated with the word must be shared among these sequences. That is, there is a set (or class) of sequences, each of which is a possible realization of the word in the utterance. This suggests a class-based method to allow for the occurrences of silence tokens. Class-based language models can be obtained by composing a language model G defined on classes with a weighted transducer T mapping classes to their members and projecting the result $G \circ T$ on the output [3]. In the current case, the input label of each successful path $\pi \in T$ will be $w\epsilon^{k+1}$ and the output label $w\langle\text{silence}\rangle^k\epsilon$ for some $k \geq 0$. If the class transducer T is pushable, and the language model G over classes is stochastic, the result is pushable.

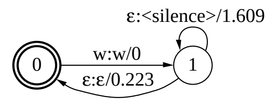

Fig. 4. A silence class transducer

Figure 4 shows one such weighted transducer over the log semiring, with a looping silence transition with probability $p = 0.2$, and with probability $1-p$ of ending the silence sequence. Our general class-based approach allows for more complex silence sequence modeling, but for the current approach we adopted a transducer with the above structure, and a single p for silences, independent of w. This results in a model that is similar to an approach for filled pauses advocated by [11]. The probability p can be estimated from a forced alignment or empirically optimized.

4.2.2. Other Sources of Non-Stochasticity

This leaves two non-stochastic knowledge source types to discuss: the compact, approximate automata representation and word or other insertion penalties. n-gram language models can be compactly encoded as finite automata by introducing backoff states that are visited by ϵ-transitions carrying the backoff weights [3]. Since these ϵ-transitions are taken freely instead of only when a match at the higher-level n-gram fails, this approximation introduces excess probability mass into the model due to multiple ways of matching. (Positive) insertion penalties, on the other hand, reduce probability mass. Since it is excess probability mass on cycles that makes automata unpushable, it is the n-gram representation that presents the problem here. There are several solutions: (1) the integrated recognition transducer can be ϵ-removed and determinized, (2) the insertion penalties can be chosen large enough to compensate for the n-gram approximation, or (3) the recognition transducer can be built without explicitly pushing it but instead constructing it nearly pushed by using log semiring operations (esp. determinization) throughout. The first gives the best results in recognition time, but can result in large transducer size increases ($10x$ is typical). The second is how we built our previously published DARPA NAB and BN tasks. The third method is, in practice, nearly as good as the first, but applicable even when the task's insertion penalties, optimized for accuracy, are small.

5. EXPERIMENTAL RESULTS

The construction algorithm described has been implemented and incorporated in the DCD Library [2]. In particular, the command-line utility `dmake` can be used to build optimized transducers. For instance, the following command can be used to build an optimized integrated recognition transducer `hclg.fsm` mapping sequences of HMM states to word sequences from a language model `g.fsm`, a lexicon transducer `l.fsm`, a context-dependency transducer `c.fsm`, and the HMM state-level transducer `h.fsm`:

```
dmake -a h.fsm -b c.fsm -a l.fsm -b g.fsm >hclg.fsm
```

The `-b` flag declares the arguments `c.fsm` and `g.fsm` to be bi-determinizable, and the `-a` flag indicates that `h.fsm` and `l.fsm` are acyclic and need to be disambiguated, closed and made determinizable as previously described. `dmake` can be used similarly with class-based language models. The following command constructs an optimized integrated transducer with a class-based language model:

```
dmake -ah.fsm -bc.fsm -al.fsm -pa tc.fsm -bgc.fsm>hclg.fsm
```

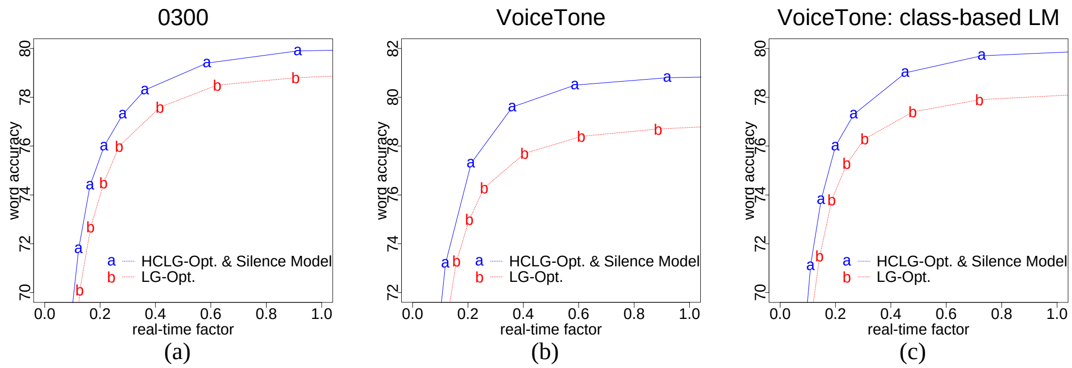

Fig. 5. Comparison of different optimization methods in (a) the 4,600-word vocabulary HMIHY 0300 task and in a 5,400-word vocabulary task of the AT&T VoiceTone project with (b) a word-based language model and (c) a class-based language model.

where `tc.fsm` is an acyclic weighted transducer mapping word sequences to classes and `gc.fsm` a language model defined on classes. The `-p` flags indicates that the result of the composition of `tc.fsm` and `gc.fsm` must be projected on the input before composition with `c.fsm`.

We used the `dmake` utility to build optimized integrated transducers for two large-vocabulary speech recognition tasks, HMIHY 0300, and a AT&T VoiceTone spoken-dialog task. The tasks share the same acoustic model consisting of 9,219 distinct HMM states each associated to a four-Gaussian mixture model, and the same triphonic context-dependency model with 2,641 states and 551,969 transitions. The language models were trigram Katz backoff models shrunk with a threshold of 0 using the method of Seymore-Rosenfeld [13], built with the utilities of the GRM Library [4]. All our recognition experiments were done using `drecog`, a general-purpose one-pass Viterbi decoder [2], on a single processor of an Intel Pentium III 1GHz linux cluster with 256KB of cache and 2GB of memory.

Figure 5(a) gives recognition accuracy as a function of recognition time for the 4,600-word vocabulary HMIHY 0300 task. The accuracy achieved at .6 times real-time by the transducer optimized at the lexicon level is achieved at .36 times real-time by the optimized transducer at the HMM-state level with the new silence model described in section 4.2, the probability mass reserved for the silences at every state of the language model being 39%. This represents a speed-up of 1.6. The use of the silence model also leads to an asymptotic improvement of the word accuracy by 1.5% absolute value.

The results obtained in a 5,400-word vocabulary task of an AT&T VoiceTone task are reported in Figure 5(b) when using word-based language model is used, and in Figure 5(c) when a class-based model is used. In the case of a word-based language model, the accuracy achieved at $.4$ times real time by the transducer optimized at the lexicon level is achieved at $.22$ times real-time by the optimized transducer at the HMM-state level with the new silence model, that is 1.8 times faster. In the case of a class-based models, the accuracy achieved at $.48$ times real time by the transducer optimized at the lexicon level is achieved at $.28$ times real-time by the optimized transducer at the HMM-state level with the new silence model, that is 1.7 times faster. In both cases, the probability mass reserved for the silences at every state of the language model is 70%. The asymptotic improvement of the accuracy due to the use of the silence model is even more important than for the previous task: slightly over 2% in both cases. Note that the performance of the word-based language model is better than that of the class-based model in this task. However, class-based models seem to lead to a better performance in some spoken-dialog classification systems, which motivated our experiments.

These results show two effects of the stochastic silence modeling technique described earlier. First, the fact that the language model is stochastic allows us to take the full benefit of our optimization algorithm, which leads to a substantial speed improvement (an HMM-state level optimized transducer with a standard language model would only have been only marginally more efficient than a lexicon-level optimized transducer). Second, the fact that the new modeling technique allows the emission of silences without disrupting the lexical dependencies leads to an asymptotic increase of the word accuracy.

6. CONCLUSION

General techniques were presented for the design of efficient large-vocabulary speech recognition transducers. The efficiency of the resulting transducers was demonstrated by experiments in several tasks with both word-based and class-based statistical language models. These techniques are incorporated in a general decoder library available for download for non-commercial use [2].

7. REFERENCES

[1] C. Allauzen and M. Mohri. Generalized Optimization Algorithm for Speech Recognition Transducers. In *Proceedings of ICASSP 2003*, Hong Kong, April 2003.

[2] C. Allauzen, M. Mohri, and M. Riley. DCD Library - Decoder Library. *http://www.research.att.com/sw/tools/dcd*, AT&T Labs - Research, 2003.

[3] C. Allauzen, M. Mohri, and B. Roark. Generalized algorithms for constructing language models. In *Proceedings of ACL 2003*, pages 40–47, 2003.

[4] C. Allauzen, M. Mohri, and B. Roark. GRM Library - Grammar Library. *http://www.research.att.com/sw/tools/grm*, AT&T Labs - Research, 2003.

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

[6] M. Mohri. Generic Epsilon-Removal and Input Epsilon-Normalization Algorithms for Weighted Transducers. *International Journal of Foundations of Computer Science*, 13(1):129–143, 2002.

[7] M. Mohri. Edit-Distance of Weighted Automata: General Definitions and Algorithms. *International Journal of Foundations of Computer Science*, (to appear), 2003.

[8] M. Mohri, F. C. N. Pereira, and M. Riley. Weighted Finite-State Transducers in Speech Recognition. *Computer Speech and Language*, 16(1):69–88, 2002.

[9] M. Mohri and M. Riley. Integrated Context-Dependent Networks in Very Large Vocabulary Speech Recognition. In *Proceedings of Eurospeech '99*, Budapest, Hungary, 1999.

[10] M. Mohri and M. Riley. A Weight Pushing Algorithm for Large Vocabulary Speech Recognition. In *Proceedings of Eurospeech '01*, Aalborg, Denmark, September 2001.

[11] J. Peters. LM Studies on Filled Pauses in Spontaneous Medical Dictation. In *Proceedings of HLT-NAACL*, 2003.

[12] A. Salomaa and M. Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Springer-Verlag: New York, 1978.

[13] K. Seymore and R. Rosenfeld. Scalable backoff language models. In *Proceedings of ICSLP*, Philadelphia, Pennsylvania, 1996.