Speech Recognition by Composition of Weighted Finite Automata

Fernando C. N. Pereira
Michael D. Riley
AT&T Labs — Research
600 Mountain Ave., Murray Hill, NJ 07974

April 20, 1996

Abstract

We present a general framework based on weighted finite automata and weighted finite-state transducers for describing and implementing speech recognizers. The framework allows us to represent uniformly the information sources and data structures used in recognition, including context-dependent units, pronunciation dictionaries, language models and lattices. Furthermore, general but efficient algorithms can used for combining information sources in actual recognizers and for optimizing their application. In particular, a single *composition* algorithm is used both to combine in advance information sources such as language models and dictionaries, and to combine acoustic observations and information sources dynamically during recognition.

1 Introduction

Many problems in speech processing can be usefully analyzed in terms of the "noisy channel" metaphor: given an observation sequence o, find which intended message w is most likely to generate that observation sequence by maximizing

$$P(w, o) = P(o|w)P(w),$$

where $P(o|w)$ characterizes the *transduction* between intended messages and observations, and $P(w)$ characterizes the message generator. More generally, the transduction between messages and observations may involve several *stages* relating successive *levels* of representation:

$$\begin{array}{r}P(s_0, s_k)=P(s_k|s_0)P(s_0)\\ P(s_k|s_0)=\sum_{s_1,\ldots,s_{k-1}} P(s_k|s_{k-1})\cdots P(s_1|s_0)\end{array} \tag{1}$$

Each s_j is a sequence of units of an appropriate representation, for instance phones or syllables in speech recognition. A straightforward but useful observation is that any such a cascade can be factored at any intermediate level

$$P(s_j|s_i) = \sum_{s_l} P(s_j|s_l)P(s_l|s_i) \tag{2}$$

For computational reasons, sums and products in (1) are often replaced by minimizations and sums of negative log probabilities, yielding the approximation

$$\begin{array}{rcl}\tilde{P}(s_0, s_k) & = & \tilde{P}(s_k|s_0) + \tilde{P}(s_0)\\ \tilde{P}(s_k|s_0) & \approx & \min_{s_1,\ldots,s_{k-1}} \sum_{1\leq j\leq k} \tilde{P}(s_j|s_{j-1})\end{array} \tag{3}$$

where $\tilde{X} = -\log X$. In this formulation, assuming the approximation is reasonable, the most likely message s_0 is the one minimizing $\tilde{P}(s_0, s_k)$.

In current speech recognition systems, a transduction stage is typically modeled by a finite-state device, for example a hidden Markov model (HMM). However, the commonalities among stages are typically not exploited, and each stage is represented and implemented by "ad hoc" means. The goal of this paper is to show that the theory of weighted rational languages and transductions can be used as a general framework for transduction cascades. Levels of representation will be modeled as weighted languages, and transduction stages will be modeled as weighted transductions.

This foundation provides a rich set of operators for combining cascade levels and stages that generalizes the standard operations on regular languages, suggests novel ways of combining models of different parts of the decoding process, and supports uniform algorithms for transduction and search throughout the cascade. Computationally, stages and levels of representation are represented as weighted finite automata, and a general automata *composition* algorithm implements the relational composition of successive stages. Automata compositions can be searched with standard best-path algorithms to find the most likely transcriptions of spoken utterances. A "lazy" implementation of composition allows search and pruning to be carried out concurrently with composition so that only the useful portions of the composition of the observations with the decoding cascade is explicitly created. Finally, finite-state minimization techniques can be used to reduce the size of cascade levels and thus improve recognition efficiency [12].

Weighted languages and transductions are generalizations of the standard notions of language and transduction in formal language theory [2, 6]. A weighted language is a mapping from strings over an alphabet to weights, while a weighted transduction is a mapping from pairs of strings over two alphabets to weights. For example, when weights represent probabilities and assuming appropriate normalization, a weighted language is just a probability distribution over strings, and a weighted transduction a conditional probability distribution between strings. The weighted *rational* languages and transducers are those that can be represented by weighted finite-state acceptors (WFSAs) and weighted finite-state transducers (WFSTs), as described in more detail in the next section. In this paper we will be concerned with the weighted rational case, although some of the theory can be profitably extended more general language classes closed under intersection with regular languages and composition with rational transductions [9, 23].

The notion of weighted rational transduction arises from the combination of two ideas in automata theory: rational transductions, used in many aspects of formal language theory [2], and weighted languages and automata, developed in pattern recognition [4, 15] and algebraic automata theory [3, 5, 8]. Ordinary (unweighted) rational transductions have been successfully applied by researchers at Xerox PARC [7] and at the University of Paris 7 [13, 14, 20, 21], among others, to several problems in language processing, including morphological analysis, dictionary compression and syntactic analysis. HMMs and probabilistic finite-state language models can be shown to be equivalent to WFSAs. In algebraic automata theory, rational series and rational transductions [8] are the algebraic counterparts of WFSAs and WFSTs and give the correct generalizations to the weighted case of the standard algebraic operations on formal languages and transductions, such as union, concatenation, intersection, restriction and composition. We believe our work is the first application of these generalizations to speech processing.

While we concentrate here on speech recognition applications, the same framework and tools have also been applied to other language processing tasks such as the segmentation of Chinese text into words [22]. We explain how a standard HMM-based recognizer can be naturally viewed as equivalent to a cascade of weighted transductions, and how the approach requires no modification to accommodate context dependencies that cross higher-level unit boundaries, for instance cross-word context-dependent models. This is an important advantage of the transduction approach over the usual, but more limited "substitution" approach used in existing to speech recognizers. Substitution replaces a symbol at a higher level by its defining language at a lower level, but, as we will argue, cannot model directly the interactions between context-dependent units at the lower level.

2 Theory

2.1 The Weight Semiring

As discussed informally in the previous section, our approach relies on associating *weights* to the strings in a language, the string pairs in a transduction and the transitions in an automaton. The operations used for weight combination should reflect the intended interpretation of the weights. For instance, if the weights of automata transitions represent transition probabilities, the weight assigned to a path should be the product of the weights of its transitions, while the weight (total probability) assigned to a set of paths with common source and destination should be the sum of the weights of the paths in the set. However, if the weights represent negative log-probabilities and we are operating under the Viterbi approximation that replaces the sum of the probabilities of alternative paths by the probability of the most probable path, path weights should be the sum of the weights of the transitions in the path and the weight assigned to a set of paths should be the minimum of the weights of the paths in the set. Both of these weight structures are special cases of *commutative semirings*, which are the basis of the general theory of weighted languages, transductions and automata [3, 5, 8].

In general, a *semiring* is a set K with two binary operations, *collection* $+_K$ and *extension* $\times_K$, such that:

- collection is associative and commutative with identity 0_K;
- extension is associative with identity 1_K;
- extension distributes over collection;
- $a \times_K 0_K = 0_K \times_K a = 0$ for any $a \in K$.

The semiring is *commutative* if extension is commutative.

Setting $K = \mathbf{R}^+$ with $+$ for collection, $\times$ for extension, 0 for 0_K and 1 for 1_K we obtain the *sum-times* semiring, which we can use to model probability calculations. Setting $K = \mathbf{R}^+ \cup \{\infty\}$ with min for collection,

$+$ for extension, ∞ for 0_K and 0 for 1_K we obtain the *min-sum* semiring, which models negative log-probabilities under the Viterbi approximation.

In general, weights represent some measure of "goodness" that we want to optimize. For instance, with probabilities we are interested in the highest weight, while the lowest weight is sought for negative log-probabilities. We thus assume a total order on weights and write $\max_x f(x)$ for the optimal value of the weight-valued function f and $\operatorname{argmax}_x f(x)$ for some x that optimizes $f(x)$. We also assume that extension and collection are monotonic with respect to the total order.

In what follows, we will assume a fixed semiring K and thus drop the subscript K in the symbols for its operations and identity elements. Unless stated otherwise, all the discussion will apply to any commutative semiring, if necessary with a total order for optimization. Some definitions and calculations involve collecting over potentially infinite sets, for instance the set of strings of a language. Clearly, collecting over an infinite set is always well-defined for *idempotent* semirings such as the min-sum semiring, in which $a + a = a\ \forall a \in K$. More generally, a *closed* semiring is one in which collecting over infinite sets is well defined. Finally, some particular cases arising in the discussion below can be shown to be well defined for the plus-times semiring under certain mild conditions on the weights assigned to strings or automata transitions [4, 8].

2.2 Weighted Transductions and Languages

In the transduction cascade (1), each stage corresponds to a mapping from input-output pairs (r, s) to probabilities $P(s|r)$. More formally, stages in the cascade will be *weighted transductions* $T : \Sigma^* \times \Gamma^* \to K$ where Σ^* and Γ^* are the sets of strings over the alphabets Σ and Γ, and K is the weight semiring. We will denote by T^{-1} the *inverse* of T defined by $T(t, s) = T(s, t)$.

The right-most step of (1) is not a transduction, but rather an information source, the language model. We will represent such sources as *weighted languages* $L : \Sigma^* \to K$.

Each transduction $S : \Sigma^* \times \Gamma^* \to K$ has two associated weighted languages, its its *first* and *second projections* $\pi_1(S) : \Sigma^* \to K$ and $\pi_2(S) : \Gamma^* \to K$, defined by

$$\begin{aligned} \pi_1(S)(s) &= \textstyle\sum_{t \in \Gamma^*} S(s,t) \\ \pi_2(S)(t) &= \textstyle\sum_{s \in \Sigma^*} S(s,t) \end{aligned}$$

Given two transductions $S : \Sigma^* \times \Gamma^* \to K$ and $T : \Gamma^* \times \Delta^* \to K$, we define their *composition* $S \circ T$ by

$$(S \circ T)(r,t) = \sum_{s \in \Gamma^*} S(r,s) \times T(s,t) \tag{4}$$

For example, if S represents $P(s_l|s_i)$ and T $P(s_j|s_l)$ in (2), $S \circ T$ represents $P(s_j|s_i)$.

A weighted transduction $S : \Sigma^* \times \Gamma^* \to K$ can be also *applied* to a weighted language $L : \Sigma^* \to K$ to yield a weighted language $S[L]$ over Γ:

$$S[L](s) = \sum_{r \in \Sigma^*} L(r) \times S(r,s) \tag{5}$$

We can also identify any weighted language L with the identity transduction restricted to L:

$$L(r, r') = \begin{cases} L(r) & \text{if } r = r' \\ 0 & \text{otherwise} \end{cases}$$

Using this identification, application is transduction composition followed by projection:

$$\begin{aligned} \pi_2(L \circ S)(s) &= \textstyle\sum_{r \in \Sigma^*} \sum_{r' \in \Sigma^*} L(r,r') \times S(r',s) \\ &= \textstyle\sum_{r \in \Sigma^*} L(r,r) \times S(r,s) \\ &= \textstyle\sum_{r \in \Sigma^*} L(r) \times S(r,s) \\ &= S[L](s) \end{aligned}$$

From now on, we will take advantage of the identification of languages with transductions and use $\circ$ to express both composition and application, often leaving implicit the projections required to extract languages from transductions. In particular, the *intersection* of two weighted languages $M, N : \Sigma^* \to K$ is given by

$$\pi_1(M \circ N)(s) = \pi_2(M \circ N)(s) = M(s) \times N(s) \tag{6}$$

It is easy to see that composition is associative, that is, the result of any transduction cascade $R_1 \circ \cdots \circ R_m$ is independent of order of application of the composition operators.

For a more concrete example, consider the transduction cascade for speech recognition depicted in Figure 1, where A is the transduction from acoustic observation sequences to phone sequences, D the transduction from phone sequences to word sequences (essentially a pronunciation dictionary)

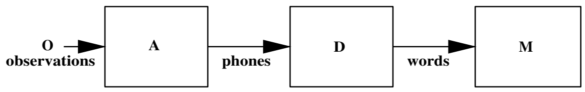

Figure 1: Recognition Cascade

	Transduction
singleton	$\{(u, v)\}(w, z) = 1$ iff $u = w$ and $v = z$
scaling	$(kT)(u, v) = k \times T(u, v)$
sum	$(S + T)(u, v) = S(u, v) + T(u, v)$
concatenation	$(ST)(t, w) = \sum_{rs=t,uv=w} S(r, u) \times T(s, v)$
power	$T^0(\epsilon, \epsilon) = 1$
	$T^0(u \neq \epsilon, v \neq \epsilon) = 0$
	$T^{n+1} = TT^n$
closure	$T^* = \sum_{k \geq 0} T^k$

Table 1: Rational Operations

and M a weighted language representing the language model. Given a particular sequence of observations o, we can represent it as the trivial weighted language O that assigns 1 to o and 0 to any other sequence. Then $O \circ A$ represents the acoustic likelihoods of possible phone sequences that generate o, $O \circ A \circ D$ the acoustic-lexical likelihoods of possible word sequences yielding o, and $O \circ A \circ D \circ M$ the combined acoustic-lexical-linguistic probabilities of word sequences generating o. The word string w with the highest weight in $\pi_2(O \circ A \circ D \circ M)$ is the most likely sentence hypothesis generating o.

Composition is thus the main operation involved in the construction and use of transduction cascades. As we will see in the next section, composition can be implemented as a suitable generalization of the usual intersection algorithm for finite automata. In addition to composition, weighted transductions (and languages, given the identification of languages with transductions presented earlier) can be constructed from simpler ones using the operations shown in Table 1, which generalize in a straightforward way the regular operations well-known from traditional automata theory [6]. In fact, the rational languages and transductions are exactly those that can be built from singletons by applications of scaling, sum, concatenation and closure.

For example, assume that for each word w in a lexicon we are given a rational transduction D_w such that $D_w(p, w)$ is the probability that w

is realized as the phone sequence p. Note that this allows for multiple pronunciations for w. Then the rational transduction $(\sum_w D_w)^*$ gives the probabilities for realizations of word sequences as phone sequences if we leave aside cross-word context dependencies, which will be discussed in Section 3.

2.3 Weighted Automata

Kleene's theorem states that regular languages are exactly those representable by finite-state acceptors [6]. Generalized to the weighted case and to transductions, it states that weighted rational languages and transductions are exactly those that can be represented by weighted finite automata [5, 8]. Furthermore, all the operations on languages and transductions we have discussed have finite-automata counterparts, which we have implemented. Any cascade representable in terms of those operations can thus be implemented directly as an appropriate combination of the programs implementing each of the operations.

A *K-weighted finite automaton* A is given by a finite set of states Q_A, a set of *transition labels* Λ_A, an initial state i_A, a *final weight* function $F_A : Q_A \to K$, [1] and a finite set $\delta_A \subset Q_A \times \Lambda_A \times K \times Q_A$ of *transitions* $t = (t.\mathrm{src}, t.\mathrm{lab}, t.\mathrm{w}, t.\mathrm{dst})$. The label set Λ_A must have with an associative *concatenation* operation $u \cdot v$ with identity element ϵ_A. A *weighted finite-state acceptor* (WFSA) is a K-weighted finite automaton with $\Lambda_A = \Sigma^*$ for some finite alphabet Σ. A *weighted finite-state transducer* (WFST) is a K-weighted finite automaton such that $\Lambda_A = \Sigma^* \times \Gamma^*$ for given finite alphabets Σ and Γ, its label concatenation is defined by $(r, s) \cdot (u, v) = (ru, sv)$, and its identity (null) label is (ϵ, ϵ). For $l = (r, s) \in \Sigma^* \times \Gamma^*$ we define $l.\mathrm{in} = r$ and $l.\mathrm{out} = s$. As we have done for languages, we will often identify a weighted acceptor with the transducer with the same state set and a transition $(q, (x, x), k, q')$ for each transition (q, x, k, q') in the acceptor.

A *path* in an automaton A is a sequence of transitions $p = t_1, \ldots, t_m$ in δ_A with $t_i.\mathrm{src} = t_{i-1}.\mathrm{dst}$ for $1 < i \leq k$. We define the *source* and the *destination* of p by $p.\mathrm{src} = t_1.\mathrm{src}$ and $p.\mathrm{dst} = t_m.\mathrm{dst}$, respectively. [2] The *label* of p is the concatenation $p.\mathrm{lab} = t_1.\mathrm{lab} \cdot \cdots \cdot t_m.\mathrm{lab}$, its *weight* is the product

[1] The usual notion of final state can be represented by $F_A(q) = 1$ if q is final, $F_A(q) = 0$ otherwise. More generally, we call a state *final* if its weight is not 0. Also, we will interpret any non-weighted automaton as a weighted automaton in which all transitions and final states have weight 1.

[2] For convenience, for each state $q \in Q_A$ we also have an *empty path* with no transitions and source and destination q.

$p.\mathrm{w} = t_1.\mathrm{w} \times \cdots \times t_m.\mathrm{w}$ and its *acceptance weight* is $F(p) = p.\mathrm{w} \times F_A(p.\mathrm{dst})$. We denote by $P_A(q, q')$ the set of all paths in A with source q and destination q', by $P_A(q)$ the set of all paths in A with source q, by $P_A^u(q, q')$ the subset of $P_A(q, q')$ with label u and by $P_A^u(q)$ the subset of $P_A(q)$ with label u.

Each state $q \in Q_A$ defines a weighted transduction (or a weighted language):

$$L_A(q)(u) = \sum_{p \in P_A^u(q)} F(p) \qquad . \tag{7}$$

Finally, we can define the weighted transduction (language) of a weighted transducer (acceptor) A by

$$[\![A]\!] = L_A(i_A) \qquad . \tag{8}$$

The appropriate generalization of Kleene's theorem to weighted acceptors and transducers states that under suitable conditions guaranteeing that the inner sum in (7) is defined, weighted rational languages and transductions are exactly those defined by weighted automata as outlined here [8].

Weighted acceptors and transducers are thus faithful implementations of rational languages and transductions, and all the operations on these described above have corresponding implementations in terms of algorithms on automata. In particular, composition is implemented by the automata operation we now describe.

2.4 Automata Composition

Informally, the composition of two automata A and B is a generalization of NFA intersection. Each state in the composition is a pair of a state of A and a state of B, and each path in the composition corresponds to a pair of a path in A and a path in B with compatible labels. The total weight of the composition path is the extension of the weights of the corresponding paths in A and B. The composition operation thus formalizes the notion of coordinated search in two graphs, where the coordination corresponds to a suitable agreement between path labels.

The more formal discussion that follows will be presented in terms of transducers, taking advantage the identifications of languages with transductions and of acceptors with transducers given earlier.

Consider two transducers A and B with $\Lambda_A = \Sigma^* \times \Gamma^*$ and $\Lambda_B = \Gamma^* \times \Delta^*$. Their composition $A \bowtie B$ will be a transducer with $\Lambda_{A \bowtie B} = \Sigma^* \times \Delta^*$ such that:

$$[\![A \bowtie B]\!] = [\![A]\!] \circ [\![B]\!] \qquad . \tag{9}$$

By definition of $L.(\cdot)$ and $\circ$ we have for any $q \in Q_A$ and $q' \in Q_B$:

$$
\begin{array}{cl}
(L_A(q) \circ L_B(q'))(u,w) & \\
= & \sum_{v \in \Gamma^*} (\sum_{p \in P_A^{(u,v)}(q)} F(p)) \times (\sum_{p' \in P_B^{(v,w)}(q')} F(p')) \\
= & \sum_{v \in \Gamma^*} \sum_{p \in P_A^{(u,v)}(q)} \sum_{p' \in P_B^{(v,w)}(q')} F(p) \times F(p') \\
= & \sum_{(p,p') \in J(q,q',u,w)} F(p) \times F(p')
\end{array}
\tag{10}
$$

where $J(q, q', u, w)$ is the set of pairs (p, p') of paths $p \in P_A(q)$ and $p' \in P_B(q')$ such that p.lab.in $= u$, p.lab.out $= p'$.lab.in and p'.lab.out $= w$. In particular, we have:

$$
([\![A]\!] \circ [\![B]\!])(u,w) = \sum_{(p,p') \in J(i_A, i_B, u, w)} F(p) \times F(p') \quad . \tag{11}
$$

Therefore, assuming that (9) is satisfied, this equation collects the weights of all paths p in A and p' in B such that p maps u to some string v and p' maps v to w. In particular, on the min-sum weight semiring, the shortest path labeled (u, w) in $[\![A \bowtie B]\!]$ minimizes the sum of the costs of paths labeled (u, v) in A and (v, w) in B, for some s.

We will give first the construction of $A \bowtie B$ for ϵ-free transducers A and B, that is, those with transition labels in $\Sigma \times \Gamma$ and $\Gamma \times \Delta$, respectively. Then $A \bowtie B$ has state set $Q_{A \bowtie B} = Q_A \times Q_B$, initial state $i_{A \bowtie B} = (i_A, i_B)$ and final weights $F_{A \bowtie B}(q, q') = F_A(q)F_B(q')$. Furthermore, there is a transition $((q, q'), (x, z), k \times k', (r, r')) \in \delta_{A \bowtie B}$ iff there are transitions $(q, (x, y), k, r) \in \delta_A$ and $(q', (y, z), k', r') \in \delta_B$. This construction is similar to the standard intersection construction for DFAs; a proof that it indeed implements transduction composition (9) is given in Appendix A.

In the general case, we consider transducers A and B with labels over $\Sigma^? \times \Gamma^?$ and $\Gamma^? \times \Delta^?$, respectively, where $\Lambda^? = \Lambda \cup \{\epsilon\}$. [3] As shown in (10), the composition of A and B should have exactly one path for each pair of paths p in A and p' in B with

$$
v = p\text{.lab.out} = p'\text{.lab.in} \quad . \tag{12}
$$

for some string $v \in \Gamma^*$ that we will call the *composition string*. In the ϵ-free case, it is clear that $p = t_1, \ldots, t_m$, $p' = t'_1, \ldots, t'_m$ for some m and t_i.lab.out $= t'_i$.lab.in. The pairing of t_i with t'_i is precisely what the ϵ-free composition construction provides. In the general case, however, two paths

[3] It is easy to see that any transducer with transition labels in $\Sigma^* \times \Gamma^*$ is equivalent to a transducer with labels in $\Gamma^? \times \Delta^?$.

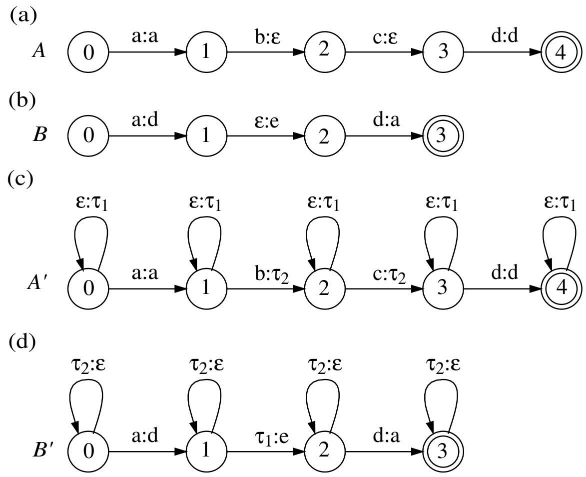

Figure 2: Transducers with ϵ Labels

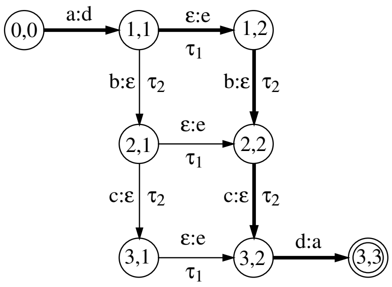

Figure 3: Composition with Marked ϵs

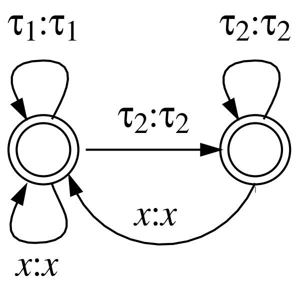

Figure 4: Filter Transducer

p and p' satisfying (12) need not have the same number of transitions. Furthermore, there may be several ways to align ϵ outputs in A and ϵ inputs in B with staying in the same state in the opposite transducer. This is exemplified by transducers A and B in Figure 2(a-b), and the corresponding naïve composition in Figure 3. The multiple paths from state $(1, 1)$ to state $(3, 2)$ correspond to different interleavings between taking the transition from 1 to 2 in B and the transitions from 1 to 2 and from 2 to 3 in A. In the weighted case, including all those paths in the composition would in general lead to an incorrect total weight for the transduction of string $abcd$ to string da. Therefore, we need a method for selecting a single composition path for each pair of compatible paths in the composed transducer.

The following construction, justified in Appendix B, achieves the desired result. For label l, define $\pi_1(l) = l.\mathrm{in}$ and $\pi_2(l) = l.\mathrm{out}$. Given a transducer T, compute $\mathrm{Mark}_i(T)$ from T by replacing the label of every transition t such that $\pi_i(t.\mathrm{lab}) = \epsilon$ with the new label l defined by $\pi_{2-i}(l) = \pi_{2-i}(t.\mathrm{lab})$ and $\pi_i(l) = \tau_i$, where τ_i is a new symbol. In words, each ϵ on the ith component of a transition label is replaced by τ_i. Corresponding to ϵ transitions on one side of the composition we need to stay in the same state on the other side. Therefore, we define the operation $\mathrm{Skip}_i(T)$ that for each state q of T adds a new transition $(q, l, 1, q)$ where $\pi_{2-i}(l) = \tau_i$ and $\pi_i(l) = \epsilon$. We also need the auxiliary transducer Filter shown in Figure 4, where the transition labeled $x : x$ is shorthand for a set of transitions mapping x to itself (at no cost) for each $x \in \Gamma$. Then for arbitrary transducers A and B, we have

$$[\![A]\!] \circ [\![B]\!] = [\![\mathrm{Skip}_1(\mathrm{Mark}_2(A)) \bowtie \mathrm{Filter} \bowtie \mathrm{Skip}_2(\mathrm{Mark}_1(B))]\!] \quad .$$

For example, with respect to Figure 2 we have $A' = \mathrm{Skip}_1(\mathrm{Mark}_2(A))$ and $B' = \mathrm{Skip}_2(\mathrm{Mark}_1(B))$. The thick path in Figure 3 is the only one allowed by the filter transduction, as desired. In practice, the substitutions and insertions of τ_i symbols performed by Mark_i and Skip_i do not need to be performed explicitly, because the effects of those operations can be computed on the fly by a suitable implementation of composition with filtering.

The filter we described is the simplest to explain. In practice, somewhat more complex filters, which we will describe elsewhere, help reduce the size of the resulting transducer. For example, the filter presented includes in the composition in states (2,1) and (3,1) on Figure 3, from which no final state can be reached. Such "dead end" paths can be a source of inefficiency when using the results of composition.

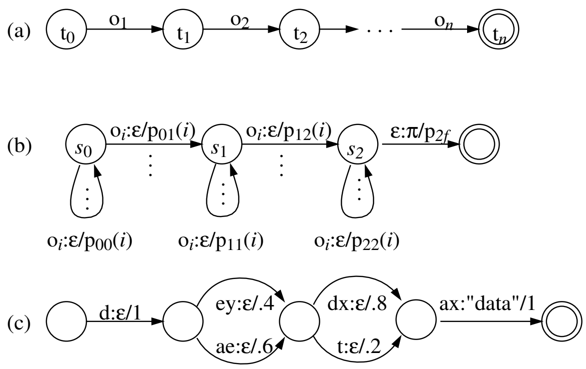

Figure 5: Models as Automata

3 Speech Recognition

We now describe how to represent a speech recognizer as a composition of transducers. Recall that we model the recognition task as the composition of a language O of acoustic observation sequences, a transduction A from acoustic observation sequences to phone sequences, a transduction D from phone sequences to word sequences and a weighted language M specifying the language model (see Figure 1). Each of these can be represented as a finite-state automaton (to some approximation), denoted by the same name as the corresponding transduction in what follows.

The acoustic observation automaton O for a given utterance has the form shown on Figure 5a. Each state represents a fixed point in time t_i, and each transition has a label, o_i, drawn from a finite alphabet that quantizes the acoustic signal between adjacent time points and is assigned probability 1. [4]

The transducer A from acoustic observation sequences to phone sequences is built from *phone models*. A phone model is a transducer from sequences of acoustic observation labels to a specific phone that assigns to each acoustic observation sequence the likelihood that the specified phone produced it. Thus, different paths through a phone model correspond to different acoustic realizations of the phone. Figure 5b shows a common topology for phone models. A is then defined as the closure of the sum of

[4]For more complex acoustic distributions (for instance, continuous densities) we can instead use multiple transitions $(t_{i-1}, d, p(o_i|d), t_i)$ where d is an observation distribution and $p(o_i|d)$ the corresponding observation probability.

the phone models.

The transducer D from phone sequences to word sequences is is built similarly to A. A *word model* is a transducer from phone sequences to the specified word that assigns to each phone sequence the likelihood that the specified word produced it. Thus, different paths through a word model correspond to different phonetic realizations of the word. Figure 5c shows a typical topology for a word model. D is then defined as the closure of the sum of the word models.

Finally, the acceptor M encodes the language model, for instance an n-gram model. Combining those automata, we obtain $\pi_2(O \bowtie A \bowtie D \bowtie M)$, which assigns a probability to each word sequence. The highest-probability path through that automaton estimates the most likely word sequence for the given utterance.

The finite-state model of speech recognition that we have just described is hardly novel. In fact, it is equivalent to that presented in [1], in the sense that it generates the same weighted language. However, the transduction cascade approach presented here allows one to view the computations in new ways.

For instance, because composition is associative, the computation of $\mathrm{argmax}_w\, \pi_2(O \bowtie A \bowtie D \bowtie M)(w)$ can be organized in a variety of ways. In a traditional integrated-search recognizer, a single large transducer is built in advance by $R = A \bowtie D \bowtie M$, and used in recognition to compute $argmax_w \pi_2(O \bowtie R)(w)$ for each observation sequence O [1]. This approach is not practical if the size of R exceeds available memory, as is typically the case for large-vocabulary speech recognition with n-gram language models for $n > 2$. In those cases, pruning may be interleaved with composition to to compute (an approximation of) $((O \bowtie A) \bowtie D) \bowtie M$. Acoustic observations are first transduced into a phone lattice represented as an automaton labeled by phones (phone recognition). The whole lattice typically too big, so the computation includes a pruning mechanism that generates only those states and transitions that appear in high-probability paths. This lattice is in turn transduced into a word lattice (word recognition), again possibly with pruning, which is then composed with the language model [11, 18]. The best approach depends on the specific task, which determines the size of intermediate results. By having a general package to manipulate weighted automata, we have been able to experiment with various alternatives.

So far, our presentation has used context-independent phone models. In other words, the likelihood assigned by a phone model in A is assumed conditionally independent of neighboring phones. Similarly, the pronunciation of each word in D is assumed independent of neighboring words. Therefore, each of the transducers has a particularly simple form, that of the closure of the sum of (inverse) *substitutions*. That is, each symbol in a string on the output side replaces a language on the input side. This replacement of a symbol from one alphabet (for example, a word) by the automaton that represents its substituted language from a over a finer-grained alphabet (for example, phones) is the usual stage-combination operation for speech recognizers [1].

However, it has been shown that context-dependent phone models, which model a phone in the context of its adjacent phones, provide substantial improvements in recognition accuracy [10]. Further, the pronunciation of a word will be affected by its neighboring words, inducing context dependencies across word boundaries.

We could include context-dependent models, such as triphone models, in our presentation by expanding our 'atomic models' in A to one for every phone in a distinct triphonic context. Each model will have the same form as in Figure 5b, but it will be over an enlarged output alphabet and have different likelihoods for the different contexts. We could also try to directly specify D in terms of the new units, but this is problematic. First, even if each word in D had only one phonetic realization, we could not directly substitute its the phones in the realization by their context-dependent models, because the given word may appear in the context of many different words, with different phones abutting the given word. This problem is commonly alleviated by either using left (right) context-independent units at the word starts (ends), which decreases the model accuracy, or by building a fully context-dependent lexicon and using special machinery in the recognizer to insure the correct models are used at word junctures. In either case, we can no longer use compact lexical entries with multiple pronunciations such as that of Figure 5c. Those approaches attempt to solve the context-dependency problem by introducing new substitutions, but substitutions are not really appropriate for the task.

In contrast, context dependency can be readily represented by a simple transducer. We leave D as defined before, but interpose a new transducer C between A and D that convert between context-dependent and context-independent units, that is, we now compute $\mathrm{argmax}_w \, \pi_2(O \bowtie A \bowtie C \bowtie D \bowtie M)(w)$. A possible form for C is shown in Figure 6. For simplicity, we show only the portion of the transducer concerning two hypothetical phones x and y. The transducer maps each context-dependent model p/l_r, associated to phone p when preceded by l and followed by r, to an occur-

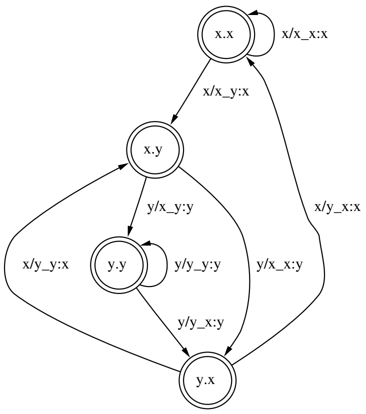

Figure 6: Context-Dependency Transducer

rence of p which is guaranteed to be preceded by l and followed by r. To ensure this, each state labeled $p.q$ represents the context information that all incoming transitions correspond to phone p, and all outgoing transitions correspond to phone q. Thus we can represent context-dependency directly as a transducer, without needing specialized context-dependency code in the recognizer. More complex forms of context dependency such as those based on classification trees over a bounded neighborhood of the target phone can too be compiled into appropriate transducers and interposed in the recognition cascade without changing any aspect of the recognition algorithm. Transducer determinization and minimization techniques [12] can be used to make context-dependency transducers as compact as possible.

4 Implementation

The transducer operations described in this paper, together with a variety of support functions, have been implemented in C. Two interfaces are provided: a library of functions operating on an abstract finite-state machine datatype, and a set of composable shell commands for fast prototyping. The modular organization of the library and shell commands follows directly from their foundation in the algebra of rational operations, and allows us to build new application-specific recognizers automatically.

The size of composed automata and the efficiency of composition have been the main issues in developing the implementation. As explained earlier, our main applications involve finding the highest-probability path in composed automata. It is in general not practical to compute the whole composition and then find the highest-probability path, because in the worst case the number of transitions in a composition grows with the product of the numbers of transitions in the composed automata. Instead, we have developed a lazy implementation of composition, in which the states and arcs of the composed automaton are created by pairing states and arcs in the composition arguments only as they are required by some other operation, such as search, on the composed automaton [19]. The use of an abstract datatype for automata facilitates this, since functions operating on automata do not need to distinguish between concrete and lazy automata.

The efficiency of composition depends crucially on the efficiency with which transitions leaving the two components of a state pair are matched to yield transitions in the composed automaton. This task is analogous to doing a relational join, and some of the sorting and indexing techniques used for joins are relevant here, especially for very large alphabets such as the words in large-vocabulary recognition. The interface of the automaton datatype has been carefully designed to allow for efficient transition matching while hiding the details of transition indexing and sorting.

5 Applications

We have used our implementation in a variety of speech recognition and language processing tasks, including continuous speech recognition in the 60,000-word ARPA North American Business News (NAB) task [18] and the 2,000-word ARPA ATIS task, isolated word recognition for directory lookup tasks, and segmentation of Chinese text into words [22].

The NAB task is by far the largest one we have attempted so far. In our 1994 experiments [18], we used a 60,000-word vocabulary, and several very large automata, including a phone-to-syllable transducer with 5×10^5 transitions, a syllable-to-word (dictionary) transducer with 10^5 transitions and a language model (5-gram) with 3.4×10^7 transitions. We are at present experimenting with various improvements in modeling and in the implementation of composition, especially in the filter, that would allow us to use directly the lazy composition of the whole decoding cascade for this application in a standard time-synchronous Viterbi decoder. In our 1994 experiments, however, we had to break the cascade into a succession of stages, each generating a pruned lattice (an acyclic acceptor) through a combination of lazy composition and graph search. In addition, relatively simple models are used first (context-independent phone models, bigram language model) to produce a relatively small pruned word lattice, which is then intersected with the composition of the full models to create a rescored lattice which is then searched for the best path. That is, we use an approximate word lattice to limit the size of the composition with the full language and phonemic models. This multi-pass decoder achieved around 10% word-error rate in the main 1994 NAB test, while requiring around 500 times real-time for recognition.

In our more recent experiments with lazy composition in synchronous Viterbi decoders, we have been able to show that lazy composition is as fast or faster than traditional methods requiring full expansion of the composed automaton in advance, while requiring a small fraction of the space. The ARPA ATIS task, for example, uses a context transducer with 40,386 transitions, a dictionary with 4,816 transitions and a class-based variable-length n-gram language model [17] with 359,532 transitions. The composition of these three automata would have around 6×10^6 transitions. However, for a typical sentence only around 5% of those transitions are actually visited [19].

6 Further Work

We have been investigating a variety of improvements, extensions and applications of the present work. With Emerald Chung, we have been refining the connection between a time-synchronous Viterbi decoder and lazy composition to improve time and space efficiency. With Mehryar Mohri, we have been developing improved composition filters, as well as exploring on-the-fly and local determinization techniques for transducers and weighted automata [12] to decrease the impact of nondeterminism on the size (and thus the time required to create) composed automata. Our work on the implementation has also been influenced by applications to the compilation of weighted phonological and morphological rules and by ongoing research on integrating speech recognition with natural-language analysis and translation. Finally, we are investigating applications to local grammatical analysis, in which transducers have been often used but not with weights.

Acknowledgments

Hiyan Alshawi, Adam Buchsbaum, Emerald Chung, Don Hindle, Andrej Ljolje, Mehryar Mohri, Steven Phillips and Richard Sproat have commented extensively on these ideas, tested many versions of our tools, and contributed a variety of improvements. Our joint work and their own separate contributions in this area will be presented elsewhere. The language model for the ATIS task was kindly supplied by Enrico Bocchieri, Roberto Pieraccini and Giuseppe Riccardi. We would also like to thank Raffaele Giancarlo, Isabelle Guyon, Carsten Lund and Yoram Singer as well as the editors of this volume for many helpful comments. Portions of this paper are adapted from a paper presented at the 1994 ARPA Human Language Technology Workshop [16].

References

[1] Lalit R. Bahl, Fred Jelinek, and Robert Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE Trans. PAMI*, 5(2):179–190, March 1983.

[2] Jean Berstel. *Transductions and Context-Free Languages*. Number 38 in Leitfäden der angewandten Mathematik and Mechanik LAMM. Teubner Studienbücher, Stuttgart, Germany, 1979.

[3] Jean Berstel and Christophe Reutenauer. *Rational Series and Their Languages*. Number 12 in EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, Germany, 1988.

[4] Taylor R. Booth and Richard A. Thompson. Applying probability measures to abstract languages. *IEEE Transactions on Computers*, C-22(5):442–450, May 1973.

[5] Samuel Eilenberg. *Automata, Languages, and Machines*, volume A. Academic Press, San Diego, California, 1974.

[6] Michael A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, Reading, Massachussets, 1978.

[7] Ronald M. Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational Linguistics*, 3(20):331–378, 1994.

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

[9] Bernard Lang. A generative view of ill-formed input processing. In *ATR Symposium on Basic Research for Telephone Interpretation*, Kyoto, Japan, December 1989.

[10] Kai-Fu Lee. Context dependent phonetic hidden Markov models for continuous speech recognition. *IEEE Trans. ASSP*, 38(4):599–609, April 1990.

[11] Andrej Ljolje and Michael D. Riley. Optimal speech recognition using phone recognition and lexical access. In *Proceedings of ICSLP*, pages 313–316, Banff, Canada, October 1992.

[12] Mehryar Mohri. On the use of sequential transducers in natural language processing. This volume.

[13] Mehryar Mohri. Compact representations by finite-state transducers. In *32nd Annual Meeting of the Association for Computational Linguistics*, San Francisco, California, 1994. New Mexico State University, Las Cruces, New Mexico, Morgan Kaufmann.

[14] Mehryar Mohri. Syntactic analysis by local grammars and automata: an efficient algorithm. In *Proceedings of the International Conference on Computational Lexicography (COMPLEX 94)*, Budapest, Hungary, 1994. Linguistic Institute, Hungarian Academy of Sciences.

[15] A. Paz. *Introduction to Probabilistic Automata.* Academic, 1971.

[16] Fernando C. N. Pereira, Michael Riley, and Richard W. Sproat. Weighted rational transductions and their application to human language processing. In *Human Language Technology Workshop*, pages 262–267, San Francisco, California, 1994. Morgan Kaufmann.

[17] Giuseppe Riccardi, Enrico Bocchieri, and Roberto Pieraccini. Nondeterministic stochastic language models for speech recognition. In *Proceedings IEE International Conference on Acoustics, Speech and Signal Processing*, volume 1, pages 237–240. IEEE, 1995.

[18] Michael Riley, Andrej Ljolje, Donald Hindle, and Fernando C. N. Pereira. The AT&T 60,000 word speech-to-text system. In J. M. Pardo, E. Enríquez, J. Ortega, J. Ferreiros, J. Macías, and F.J.Valverde, editors, *Eurospeech'95: ESCA 4th European Conference on Speech Communication and Technology*, volume 1, pages 207–210, Madrid,

Spain, September 1995. European Speech Communication Association (ESCA).

[19] Michael Riley, Fernando Pereira, and Emerald Chung. Lazy transducer composition: a flexible method for on-the-fly expansion of context-dependent grammar network. IEEE Automatic Speech Recognition Workshop, Snowbird, Utah, December 1995.

[20] Emmanuel Roche. *Analyse Syntaxique Transformationelle du Français par Transducteurs et Lexique-Grammaire*. PhD thesis, Université Paris 7, 1993.

[21] Max Silberztein. *Dictionnaires électroniques et analise automatique de textes: le système INTEX*. Masson, Paris, France, 1993.

[22] Richard Sproat, Chilin Shih, Wiliam Gale, and Nancy Chang. A stochastic finite-state word-segmentation algorithm for Chinese. In *32nd Annual Meeting of the Association for Computational Linguistics*, pages 66–73, San Francisco, California, 1994. New Mexico State University, Las Cruces, New Mexico, Morgan Kaufmann.

[23] Ray Teitelbaum. Context-free error analysis by evaluation of algebraic power series. In *Proc. Fifth Annual ACM Symposium on Theory of Computing*, pages 196–199, Austin, Texas, 1973.

A Correctness of ϵ-Free Composition

As shown in Section 2.4 (10), we have

$$(L_A(q) \circ L_B(q'))(r,t) = \sum_{s\in\Gamma^*} \sum_{p\in P_A^{(r,s)}(q)} \sum_{p'\in P_B^{(s,t)}(q')} F(p) \times F(p') \qquad .(13)$$

Clearly, for ϵ-free transducers the variables r, s, t, p and p' in this equation satisfy the constraint $|r| = |s| = |t| = |p| = |p'| = n$ for some n. This allows us to show the correctness of the composition construction for ϵ-free automata by induction on n. Specifically, we shall show that for any $q \in Q_A$ and $q' \in Q_B$

$$L_{A\bowtie B}(q,q') = L_A(q) \circ L_B(q') \qquad . \qquad (14)$$

For $n = 0$, from (13) and the composition construction we obtain

$$\begin{aligned}(L_A(q) \circ L_B(q'))(\epsilon, \epsilon) &= F_A(q) \times F_B(q') \\ &= F_{A \bowtie B}(q, q') \\ &= F_{A \bowtie B}(\epsilon, \epsilon)\end{aligned}$$

as needed.

Assume now that $L_{A \bowtie B}(m, m')(u, w) = (L_A(m) \circ L_B(m'))(u, w)$ for any $m \in Q_A$, $m' \in Q_B$, $u \in \Sigma^*$ and $w \in \Delta^*$ with $|u| = |w| < n$. Let $r = xu$ and $t = zw$, with $x \in \Sigma$ and $z \in \Delta$. Then by (13) and the composition construction we have

$$\begin{aligned}&(L_A(p) \circ L_B(q))(xu, zw) \\ &= \textstyle\sum_{y \in \Gamma} \sum_{v \in \Gamma^*} \sum_{p \in P_A^{(xu,yv)}(q)} \sum_{p' \in P_B^{(yv,zw)}(q')} F(p) \times F(p') \\ &= \sum_{(q,(x,y),k,m) \in \delta_A} \sum_{(q',(y,z),k',m') \in \delta_B} \\ &\qquad k \times k' \times (\textstyle\sum_{v \in \Gamma^*} \sum_{l \in P_A^{(u,v)}(m)} \sum_{l' \in P_B^{(v,w)}(m')} F(l) \times F(l')) \\ &= \sum_{((q,q'),(x,z),j,(m,m')) \in \delta_{A \bowtie B}} \\ &\qquad j \times (\textstyle\sum_{v \in \Gamma^*} \sum_{l \in P_A^{(u,v)}(m)} \sum_{l' \in P_B^{(v,w)}(m')} F(l) \times F(l')) \\ &= \textstyle\sum_{((q,q'),(x,z),j,(m,m')) \in \delta_{A \bowtie B}} j \times (L_A(m) \circ L_B(m'))(u, w) \\ &= \textstyle\sum_{((q,q'),(x,z),j,(m,m')) \in \delta_{A \bowtie B}} j \times L_{A \bowtie B}(m, m')(u, w) \\ &= \textstyle\sum_{((q,q'),(x,z),j,(m,m')) \in \delta_{A \bowtie B}} j \times (\sum_{g \in P_{A \bowtie B}^{(u,w)}(m,m')} W_{A \bowtie B}(g)) \\ &= \textstyle\sum_{h \in P_{A \bowtie B}^{(xu,zw)}(q,q')} W_{A \bowtie B}(h) \\ &= L_{A \bowtie B}(q, q')(xu, zw) \qquad .\end{aligned}$$

This shows (14) for ϵ-free transducers, and as a particular case

$$[\![A \bowtie B]\!] = [\![A]\!] \circ [\![B]\!] \qquad ,$$

which states that transducer composition correctly implements transduction composition.

B General Composition Construction

For any transition t in A or B, we define

$$\mathrm{Mark}_i(t) = \begin{cases} \tau_i & \text{if } \pi_i(t.\mathrm{lab}) = \epsilon \\ \pi_i(t.\mathrm{lab}) & \text{otherwise} \end{cases} \qquad ,$$

where each τ_i is a new symbol not in Γ. This can be extended to a path $p = t_1, \ldots, t_m$ in the obvious way by $\text{Mark}_i(p) = \text{Mark}_i(t_1) \cdots \text{Mark}_i(t_m)$. If p and p' satisfy (12), there will be $m, n \geq k$ such that $p = t_1, \ldots, t_m$, $p' = t'_1, \ldots, t'_n$, $v = y_1 \cdots y_k$ and $v = p.\text{lab.out} = p'.\text{lab.in}$. Therefore, we will have $\text{Mark}_2(p) = u_0 y_1 u_1 \cdots u_{k-1} y_k u_k$ where $u_i \in \{\tau_2\}^*$ and $|u_0 \cdots u_k| = m - k$, and $\text{Mark}_1(p') = v_0 y_1 v_1 \cdots v_{k-1} y_k v_k$ where $v_i \in \{\tau_1\}^*$ and $|v_0 \cdots v_k| = n - k$.

We will need the following standard definition of the *shuffle* $s \star s'$ of two languages $L, L' \subseteq \Gamma^*$:

$$L \star L' = \{u_1 v_1 \cdots u_l v_l | u_1 \cdots u_l \in L, v_1 \cdots v_l \in L'\} \qquad .$$

Then it is easy to see that (12) holds iff

$$J = (\{\text{Mark}_2(p)\} \star \{\tau_1\}^*) \cap (\{\text{Mark}_1(p')\} \star \{\tau_2\}^*) \neq \emptyset \qquad . \tag{15}$$

Each composition string $v \in J$ has the form

$$v = v_0 y_1 v_1 \cdots v_{k-1} y_k v_k \tag{16}$$

for $y_i \in \Gamma$ and $v_i \in \{\tau_1, \tau_2\}^*$. Furthermore, by construction, any string $v'_0 y_1 v'_1 \cdots v'_{k-1} y_k v'_k$, where each v'_i is derived from v_i by commuting τ_1 instances with τ_2 instances, is also in J.

Consider for example the transducers A shown in Figure 2a and B shown in Figure 2b. For path p from state 0 to state 4 in A and path p' from state 0 to state 3 in B we have the following equalities:

$$\begin{array}{rcl} \text{Mark}_2(p) & = & a\tau_2\tau_2 d \\ \text{Mark}_1(p') & = & a\tau_1 d \\ (\{\text{Mark}_2(p)\} \star \{\tau_1\}^*) \cap (\{\text{Mark}_1(p')\} \star \{\tau_2\}^*) & = & \left\{ \begin{array}{l} a\tau_1\tau_2\tau_2 d, \\ a\tau_2\tau_1\tau_2 d, \\ a\tau_2\tau_2\tau_1 d \end{array} \right\} \end{array}$$

Therefore, p and p' satisfy (12), allowing $[\![A]\!] \circ [\![B]\!]$ to map *abcd* to *dea*. It is also straightforward to see that, given the transducers A' in Figure 2c and B' in Figure 2d, we have

$$\begin{array}{rcl} \{\text{Mark}_2(p)\} \star \{\tau_1\}^* & = & \{p.\text{lab.out} | p \in P_{A'}(0)\} \\ \{\text{Mark}_1(p')\} \star \{\tau_2\}^* & = & \{p'.\text{lab.in} | p' \in P_{B'}(0)\} \end{array}$$

Since there are no ϵ labels on the output side of A' or the input side of B', we can apply to them the ϵ-free composition construction, with the result shown in Figure 3. Each of the paths from the initial state to the final state corresponds to a different composition string in $\{\mathrm{Mark}_2(p)\} \star \{\tau_1\}^* \cap \{\mathrm{Mark}_1(p')\} \star \{\tau_2\}^*$.

The transducer $A' \bowtie B'$ pairs up exactly the strings it should, but it does not correctly implement $[\![A]\!] \circ [\![B]\!]$ in the general weighted case. The construction described so far allows several paths in $A' \bowtie B'$ corresponding to each pair of paths from A and B. Intuitively, this is possible because τ_1 and τ_2 are allowed to commute freely in the composition string. But if one pair of paths p from A and p' from B leads to several paths in $A' \bowtie B'$, the weights from the ϵ-transitions in A and B will appear multiple times in the overall weight for going from $(p.\mathrm{src}, p'.\mathrm{src})$ to $(p.\mathrm{dst}, p'.\mathrm{dst})$ in $A' \bowtie B'$. If the semiring sum operation is not idempotent, that leads to the wrong weights in (10).

To achieve the correct path multiplicity, we interpose a transducer Filter between A' and B' in a 3-way composition $\bowtie (A', \mathrm{Filter}, B')$. The Filter transducer is shown in Figure 4, where the transition labeled $x : x$ represents a set of transitions mapping x to itself for each $x \in \Gamma$. The effect of Filter is to block any paths in $A' \bowtie B'$ corresponding to a composition string containing the substring $\tau_2\tau_1$. This eliminates all the composition strings (16) in (15) except for the one with $v_i \in \{\tau_1\}^*\{\tau_2\}^*$, which is guaranteed to exist since J in (15) allows all interleavings of τ_1 and τ_2, including the required one in which all τ_2 instances must follow all τ_1 instances. For example, Filter would remove all but the thick-lines path in Figure 3, as needed to avoid incorrect path multiplicities.