Filters for Efficient Composition of Weighted Finite-State Transducers

Cyril Allauzen, Michael Riley, and Johan Schalkwyk

Google Research, 76 Ninth Avenue, New York, NY 10011, USA
{allauzen,riley,johans}@google.com

Abstract. This paper describes a weighted finite-state transducer composition algorithm that generalizes the concept of the *composition filter* and presents various filters that process epsilon transitions, look-ahead along paths, and push forward labels along epsilon paths. These filters, either individually or in combination, make it possible to compose some transducers much more efficiently in time and space than otherwise possible. We present examples of this drawn, in part, from demanding speech-processing applications. The generalized composition algorithm and many of these filters have been included in *OpenFst*, an open-source weighted transducer library.

1 Introduction

The *composition* algorithm plays a central role in the use of weighted finite-state transducers. It is used, for example, to apply finite-state models to inputs and to combine cascaded models. The classical version of the composition algorithm, which simply matches transitions leaving paired input states, is easy to implement and often effective in practice. However, experience has shown that there are some transducers of practical importance that do not compose efficiently in this way. These cases typically create significant numbers of non-coaccessible composition states that waste time and space. For some problems, it is possible to find equivalent inputs that will compose more efficiently, but it is not always possible or desirable to do so. This has been especially an issue in natural language processing applications and led to special-purpose composition algorithms for use in speech recognition [5, 6, 10, 14] and speech synthesis [2].

In this paper we generalize the composition algorithm, subsuming several of these specializations and others in an efficient way. The idea is to introduce a composition *filter*, applied at each composition state during the construction, that decides if composition is to continue. If we set out to create a general composition filter that blocks every non-coaccessible composition state for any input transducers, then we have only delegated the job of doing a full composition to the filter. Instead, we take the view that there are certain specific filters, tailored to particular but common cases, that are efficient to use, involving only a limited degree of look-ahead along paths. Composition itself is then parameterized to take one or more of these filters that are selected by the user to fit his problem.

Section 2 presents the generalized composition algorithm and defines several composition filters. Section 3 provides examples of these composition filters applied to practical problems. Section 4 briefly describes how these filters are used in *OpenFst* [3], an open-source weighted transducer library.

2 Composition Algorithm

2.1 Preliminaries

A semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is ring that may lack negation. If $\otimes$ is commutative, we say that the semiring is *commutative.*

The *probability semiring* $(\mathbb{R}_+, +, \times, 0, 1)$ is used when the weights represent probabilities. The *log semiring* $(\mathbb{R} \cup \{\infty\}, \oplus_{\log}, +, \infty, 0)$, isomorphic to the probability semiring via the negative-log mapping, is often used in practice for numerical stability. The *tropical semiring* $(\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0)$, derived from the log semiring using the *Viterbi approximation*, is often used in shortest-path applications.

A *weighted finite-state transducer* $T = (\mathcal{A}, \mathcal{B}, Q, I, F, E, \lambda, \rho)$ over a semiring $\mathbb{K}$ is specified by a finite input alphabet $\mathcal{A}$, a finite output alphabet $\mathcal{B}$, a finite set of states Q, a set of initial states $I \subseteq Q$, a set of final states $F \subseteq Q$, a finite set of transitions $E \subseteq \overline{E} = Q \times (\mathcal{A} \cup \{\epsilon\}) \times (\mathcal{B} \cup \{\epsilon\}) \times \mathbb{K} \times Q$, an initial state weight assignment $\lambda : I \to \mathbb{K}$, and a final state weight assignment $\rho : F \to \mathbb{K}$. $E[q]$ denotes the set of transitions leaving state $q \in Q$.

Given a transition $e \in E$, $p[e]$ denotes its origin or previous state, $n[e]$ its destination or next state, $i[e]$ its input label, $o[e]$ its output label, and $w[e]$ its weight. A *path* $\pi = e_1 \cdots e_k$ is a sequence of consecutive transitions: $n[e_{i-1}] = p[e_i]$, $i = 2, \ldots, k$. The functions n, p, and w on transitions can be extended to paths by setting: $n[\pi] = n[e_k]$ and $p[\pi] = p[e_1]$ and by defining the weight of a path as the $\otimes$-product of the weights of its constituent transitions: $w[\pi] = w[e_1] \otimes \cdots \otimes w[e_k]$. A *string* is a sequence of labels; ϵ denotes the empty string.

The weight associated by T to any pair of input-output strings (x, y) is given by:

$$T(x,y) = \bigoplus_{\pi \in \cup_{q \in I,\, q' \in F} P(q,x,y,q')} \lambda[p[\pi]] \otimes w[\pi] \otimes \rho[n[\pi]], \tag{1}$$

where $P(q, x, y, q')$ denotes the set of paths from q to q' with input label $x \in \mathcal{A}^*$ and output label $y \in \mathcal{B}^*$.

We denote by $|T|_Q$ the number of states, $|T|_E$ the number of transitions, and $d(T)$ the maximum out-degree in T. The *size* of T is then $|T| = |T|_Q + |T|_E$.

2.2 Composition

Let $\mathbb{K}$ be a commutative semiring and let T_1 and T_2 be two weighted transducers defined over $\mathbb{K}$ such that the input alphabet $\mathcal{B}$ of T_2 coincides with the output alphabet of T_1. The result of the composition of T_1 and T_2 is a weighted transducer denoted by $T_1 \circ T_2$ and specified for all x, y by:

$$(T_1 \circ T_2)(x,y) = \bigoplus_{z \in \mathcal{B}^*} T_1(x,z) \otimes T_2(z,y). \tag{2}$$

Leaving aside transitions with ϵ inputs or outputs, the following rule specifies how to compute a transition of $T_1 \circ T_2$ from appropriate transitions of T_1 and T_2: (q_1, a, b, w_1, q_1') and (q_2, b, c, w_2, q_2') results in $((q_1, q_2), a, c, w_1 \otimes w_2, (q_1', q_2'))$. A simple algorithm to compute the composition of two ϵ-free transducers, following the above rule, is given in [13].

More care is needed when T_1 has output ϵ labels or T_2 input ϵ labels. An output ϵ label in T_1 may be matched with an input ϵ label in T_2, following the above rule with ϵ labels treated as regular symbols. However, an output ϵ label may also be read in T_1 without matching any actual transition in T_2. This case can be handled by the above rule after adding self-loops at every state of T_2 labeled on the inner tape by a new symbol ϵ^L and on the outer tape by ϵ and allowing transitions labeled by ϵ and ϵ^L to match. Similar self-loops are added to T_1 for matching input ϵ labels on T_2. However, this approach can result in redundant ϵ-paths since an epsilon label can match in the two above ways. The redundant paths must be *filtered* out because they will produce incorrect results in non-idempotent semirings (like the log semiring).[1] We introduced the ϵ^L label to distinguish these two types of match in the filtering.

In [13], a *filter transducer* is introduced that is used with relabeling and the ϵ-free composition algorithm to correctly implement composition with ϵ labels. Our composition algorithm extends this by generalizing the *composition filter*.

Our algorithm takes as input two weighted transducers $T_1 = (\mathcal{A}, \mathcal{B}, Q_1, I_1, F_1, E_1, \lambda_1, \rho_1)$ and $T_2 = (\mathcal{B}, \mathcal{C}, Q_2, I_2, F_2, E_2, \lambda_2, \rho_2)$ over a semiring $\mathbb{K}$ and a composition filter $\Phi = (T_1, T_2, Q_3, i_3, \perp, \varphi, \rho_3)$, which has a set of filter states Q_3, a designated initial filter state i_3, a designated blocking filter state $\perp$, a transition filter $\varphi : E_1^L \times E_2^L \times Q_3 \to \overline{E}_1 \times \overline{E}_2 \times Q_3$ where $E_n^L = \bigcup_{q \in Q_n} E^L[q]$, $E^L[q_1] = E[q_1] \cup \big\{(q_1, \epsilon, \epsilon^L, \overline{1}, q_1)\big\}$ for each $q_1 \in Q_1$, $E^L[q_2] = E[q_2] \cup \big\{(q_2, \epsilon^L, \epsilon, \overline{1}, q_2)\big\}$ for each $q_2 \in Q_2$ and a final weight filter $\rho_3 : Q_3 \to \mathbb{K}$.

We shall see that the filter can be used in composition to block the expansion of some states (by entering the $\perp$ state) and modify the transitions and final weights (useful for optimizations).

The states in the output of composition are identified with triples of a state from each of the two input transducers and one from the filter. In particular, the algorithm outputs a weighted finite-state transducer $T = (\mathcal{A}, \mathcal{C}, Q, I, F, E, \lambda, \rho)$ implementing the composition of T_1 and T_2 where $Q \subseteq Q_1 \times Q_2 \times Q_3$ and $I = I_1 \times I_2 \times \{i_3\}$.

Figure 1 gives the pseudocode of this algorithm. E and F are all initialized to the empty set and grown as needed. The algorithm uses a queue S containing the set of state triples of states yet to be examined. The queue discipline of S is arbitrary and does not affect the termination of the algorithm. The state set Q is initially the set of triples of initial states of the original transducers and filter, as is I and S, and the corresponding initial weights are computed (lines

[1] Redundant ϵ-paths are also an issue in the unweighted case when testing for the ambiguity of finite automata [1].

```
Weighted-Composition($T_1, T_2, \Phi$)
 1  $Q \leftarrow I \leftarrow S \leftarrow I_1 \times I_2 \times \{i_3\}$
 2  for each $(q_1, q_2, i_3) \in I$ do
 3      $\lambda(q_1, q_2, i_3) \leftarrow \lambda_1(q_1) \otimes \lambda_2(q_2)$
 4  while $S \neq \emptyset$ do
 5      $(q_1, q_2, q_3) \leftarrow$ Head($S$)
 6      Dequeue($S$)
 7      if $(q_1, q_2, q_3) \in F_1 \times F_2 \times Q_3$ and $\rho_3(q_3) \neq \overline{0}$ then
 8          $F \leftarrow F \cup \{(q_1, q_2, q_3)\}$
 9          $\rho(q_1, q_2, q_3) \leftarrow \rho_1(q_1) \otimes \rho_2(q_2) \otimes \rho_3(q_3)$
10      $M \leftarrow \{(e_1, e_2) \in E^L[q_1] \times E^L[q_2]$ s.t. $\varphi(e_1, e_2, q_3) = (e'_1, e'_2, q'_3)$ with $q'_3 \neq \perp\}$
11      for each $(e_1, e_2) \in M$ do
12          $(e'_1, e'_2, q'_3) \leftarrow \varphi(e_1, e_2, q_3)$
13          if $(n[e'_1], n[e'_2], q'_3) \notin Q$ then
14              $Q \leftarrow Q \cup \{(n[e'_1], n[e'_2], q'_3)\}$
15              Enqueue($S, (n[e'_1], n[e'_2], q'_3)$)
16          $E \leftarrow E \cup \{((q_1, q_2, q_3), i[e'_1], o[e'_2], w[e'_1] \otimes w[e'_2], (n[e'_1], n[e'_2], q'_3))\}$
17  return $T$
```

Fig. 1. Pseudocode of the composition algorithm.

1-3). Each time through the loop in lines 3-14, a new triple of states (q_1, q_2, q_3) is extracted from S (lines 5-6). The final weight of (q_1, q_2, q_3) is computed by $\otimes$-multiplying the final weights of q_1 and q_2 and the final filter weight when they are all final states (lines 8-9). Then, for each pair of transitions, the transition filter is first applied. If the new filter state is not the blocking state $\perp$ and a new transition is created from the filter-rewritten transitions (e'_1, e'_2) (line 16). If the destination state $(n[e'_1], n[e'_2], q'_3)$ has not been found previously, it is added to Q and inserted in S (lines 13-15). The composition algorithm presented here is available in the *OpenFst* library [3].

2.3 Elementary Composition Filters

In this section, we consider elementary filters for composition without and with epsilon transitions.

Trivial Filter Filter Φ_{trivial} blocks no paths and leaves transitions and final weights unmodified. For Φ_{trivial}, let $Q_3 = \{0, \perp\}$, $i_3 = 0$, $\varphi(e_1, e_2, q_3) = (e_1, e_2, q'_3)$ with $q'_3 = 0$ if $o[e_1] = i[e_2] \in \mathcal{B}$ and $\perp$ otherwise, and $\rho(q_3) = \overline{1}$ for all $q_3 \in Q_3$. With this filter, the pseudocode in Figure 1 matches the simple epsilon-free composition algorithm given in [13].

Let us assume that the transitions at each state in T_2 are sorted according to their input label. The set M of transitions to be computed line 8 is simply equal to $\{(e_1, e_2) \in E[q_1] \times E[q_2] : o[e_1] = i[e_2]\}$. It can be computed by performing a binary search over $E[q_2]$ for each transition in $E[q_1]$. The time complexity of computing M is then $O(|E[q_1]| \log |E[q_2]| + |M|)$. Since each element in M will result in a transition in T, the worst-case time complexity of the algorithm is $O(|T|_Q d(T_1) \log d(T_2) + |T|_E)$. The space complexity of the algorithm is $O(|T|)$.

Epsilon-Matching Filter Filter $\Phi_{\epsilon\text{-match}}$ handles epsilon labels, but disallows redundant epsilon paths, preferring those that match actual ϵ labels. It leaves transitions and final weights unmodified.

For $\Phi_{\epsilon\text{-match}}$, let $Q_3 = \{0, 1, 2, \perp\}$, $i_3 = 0$, $\rho(q_3) = \overline{1}$ for all $q_3 \in Q_3$, and $\varphi(e_1, e_2, q_3) = (e_1, e_2, q_3')$ where:

$$q_3' = \begin{cases} 0 \text{ if } (o[e_1], i[e_2]) = (x, x) \text{ with } x \in \mathcal{B}, \\ 0 \text{ if } (o[e_1], i[e_2]) = (\epsilon, \epsilon) \text{ and } q_3 = 0, \\ 1 \text{ if } (o[e_1], i[e_2]) = (\epsilon^L, \epsilon) \text{ and } q_3 \neq 2, \\ 2 \text{ if } (o[e_1], i[e_2]) = (\epsilon, \epsilon^L) \text{ and } q_3 \neq 1, \\ \perp \text{ otherwise.} \end{cases}$$

With this filter, the pseudocode in Figure 1 matches the composition algorithm given in [13] with the specified composition filter transducer. The complexity of the algorithm is the same as when using the trivial filter.

Epsilon-Sequencing Filter Alternatively, filter $\Phi_{\epsilon\text{-seq}}$ can also be used to remove redundant epsilon paths. This filter favors epsilon paths consisting of (output) ϵ-transitions in T_1 (matched with staying at the same state in T_2) followed by (input) ϵ-transitions in T_2 (matched with staying at the same state in T_1).

For $\Phi_{\epsilon\text{-seq}}$, let $Q_3 = \{0, 1, \perp\}$, $i_3 = 0$, $\rho(q_3) = \overline{1}$ for all $q_3 \in Q_3$, and $\varphi(e_1, e_2, q_3) = (e_1, e_2, q_3')$ where:

$$q_3' = \begin{cases} 0 \text{ if } (o[e_1], i[e_2]) = (x, x) \text{ with } x \in \mathcal{B}, \\ 0 \text{ if } (o[e_1], i[e_2]) = (\epsilon, \epsilon^L) \text{ and } q_3 = 0, \\ 1 \text{ if } (o[e_1], i[e_2]) = (\epsilon^L, \epsilon), \\ \perp \text{ otherwise.} \end{cases} \tag{3}$$

The complexity of the algorithm is the same as when using the trivial filter. Replacing the pair $(o[e_1], i[e_2])$ by $(i[e_2], o[e_1])$ in (3) leads to the symmetric filter $\overline{\Phi}_{\epsilon\text{-seq}}$. Whether it is better to choose the epsilon-matching or epsilon-sequencing filter is problem-dependent as shown in Section 3.

2.4 Look-Ahead Composition Filters

In this section, we introduce filters that can result in more efficient composition by looking-ahead along paths and blocking unsuccessful matches under various scenarios.

String-Potential Filter Filter Φ_{sp} looks-ahead along common prefixes of state futures. Given two strings u and v, we denote by $u \wedge v$ the longest common prefix of u and v. Given a state q in a tranducer T, the input (resp. output) string potential of q, denoted by $p_i(q)$ (resp. $p_o(q)$), is the longest common prefix of the input (resp. output) labels of all the paths from q to a final state.

For Φ_{sp}, let $Q_3 = \{0, \bot\}$, $i_3 = 0$, $\rho(0) = \overline{1}$, and $\varphi(e_1, e_2, q_3) = (e_1, e_2, q_3')$ where:

$$q_3' = \begin{cases} 0 \ \text{ if } p_o(n[e_1]) \wedge p_i(n[e_2]) \in \{p_o(n[e_1]), p_i(n[e_2])\}, \\ \bot \text{ otherwise.} \end{cases}$$

This filter prevents the creation of some non-coaccessible states since a state (q_1, q_2) in $T_1 \circ T_2$ is coaccessible only if $p_o(q_1)$ is a prefix of $p_i(q_2)$ or $p_i(q_2)$ is a prefix of $p_o(q_1)$ [2]. Computing string potentials can be done using the generic single-source shortest-distance algorithm of [12] over the string semiring. This can be done on-demand or as a pre-processing step. Naively storing a string at each state results in a complexity (on-demand) of $O(|T|_Q d(T_1) \log d(T_2) + |T|_E \min(\mu_1, \mu_2))$ in time and $O(|T| + |T_1|_Q \mu_1 + |T_2|_Q \mu_2)$ in space, with μ_i being the length of the longest potential in T_i. This can be improved using better data structures (such as tries or suffix trees).

Transition-Look-Ahead Filter When states paired in composition have no shared common prefixes, it is is necessary to examine the specific transitions themselves in any look-ahead. A simple form of look-ahead is then to try to match one set of transitions into the future.

Given a state q in a transducer T let us denote by $L_i(q)$ and $L_o(q)$ the set of input and output labels of outgoing transitions in q. For $\Phi_{\text{tr-la}}$, let $Q_3 = \{0, \bot\}$, $i_3 = 0$, $\rho(0) = \overline{1}$, and $\varphi(e_1, e_2, q_3) = (e_1, e_2, q_3')$ where:

$$q_3' = \begin{cases} 0 \ \text{ if } L_o(n[e_1]) \cap L_i(n[e_2]) \neq \emptyset \text{ or } \epsilon \in L_o(n[e_1]) \cup L_i(n[e_2]), \\ \bot \text{ otherwise.} \end{cases}$$

The sets $L_i(q)$ and $L_o(q)$ can be computed on-demand or as a pre-processing step and can be represented using data-structures providing efficient intersection such as bit vectors or Bloom filters. Using bit vectors, the complexity (on-demand) is $O(|T|_Q d(T_1) \log d(T_2) + |T|_E \log |\mathcal{B}|)$ in time and $O(|T| + (|T_1|_Q + |T_2|_Q) \log |\mathcal{B}|)$ in space.

Label-Reachability Filter In transducers with epsilon transitions, looking-ahead a single transition is not sufficient, since we can not match a (non-epsilon) label without traversing epsilon paths. Filter Φ_{reach} precomputes those traverals.

When composing states q_1 in T_1 and q_2 in T_2, filter Φ_{reach} disallows following an epsilon-labeled path from q_1 that will fail to reach a non-epsilon label that matches some transition leaving state q_2. It leaves transitions and final weights unmodified. For simplicity, we assume there are no input ϵ labels in T_1.

For Φ_{reach}, let $Q_3 = \{0, \bot\}$, $i_3 = 0$, and $\rho(q_3) = \overline{1}$ for all $q_3 \in Q_3$. Define $r : \mathcal{B} \times Q_1 \to \{0, 1\}$ such that $r(x, q) = 1$ if there is a path π from q to some q' in T_1 with $o[\pi] = x$, otherwise let $r(x, q) = 0$. Let $\varphi(e_1, e_2, q_3) = (e_1, e_2, 0)$ if (i) $o[e_1] = i[e_2]$ or if (ii) $o[e_1] = \epsilon, i[e_2] = \epsilon^L$, and for some $e_2' \in E[p[e_2]]$, $i[e_2'] \neq \epsilon$ and $r(i[e_2'], n[e_1]) = 1$. Otherwise let $\varphi(e_1, e_2, q_3) = (e_1, e_2, \bot)$.

Let us denote by $c_r(T_1)$ the cost of performing one reachability query in T_1 using r, by $S_r(T_1)$ the total space required for r, and by $d_\epsilon T_1$ the maximal number of output-ϵ transitions at a state in T_1. The worst-case time complexity of the algorithm is: $O(|T|_Q(d(T_1)\log d(T_2)+d_\epsilon(T_1)c_r(T_1))+|T|_E)$, and the space complexity is $O(|T|+S_r(T_1))$.

There are different ways we can represent r and they will lead to different complexities for composition. We will assume for our analysis, whatever its representation, that r is precomputed and stored with T_1. In general, we exclude any T-specific precomputation from composition's time complexity.

Point Representation of r: Define $R_q = \{x \in \mathcal{B} : r(x,q) = 1\}$ for each state $q \in T_1$. If the labels in R_q are stored in a linked list, traversed linearly and each matched against sorted input labels in T_2 using binary search, then $c_r(T_1) = \max_q |R_q| \log d(T_2)$ and $S_r(T_1) = \sum_q |R_q|$.

Interval Representation of r: We can use intervals to represent R_q if $\mathcal{B} = [1, |\mathcal{B}|] \subset \mathbb{N}$ by defining $I_q = \{[x,y) : x,y \in \mathbb{N}, [x,y) \subseteq R_q, x-1 \notin R_q, y \notin R_q\}$. If the intervals in I_q are stored in a linked list, traversed linearly and each matched against sorted input labels in T_2 using (lower-bound) binary search, then $c_r(T_1) = \max_q |I_q| \log d(T_2)$ and $S_r(T_1) = \sum_q |I_q|$.

Assuming the particular numbering of the labels is arbitrary, let permutation $\Pi : \mathcal{B} \to \mathcal{B}$ be a bijection that is used to relabel both T_1 and T_2 prior to composition. Among the $|\mathcal{B}|!$ different possible such permutations, some could result in far fewer intervals in I_q than others. In fact, there may exist a Π that results in one interval per I_q. Consider the $|\mathcal{B}| \times |Q_1|$ matrix $\mathbf{R}$ with $\mathbf{R}[i,j] = r(i,j)$. The condition that the I_q each contain a single interval is equivalent to the property that the ones in the columns of $\mathbf{R}$ are consecutive. A binary matrix $\mathbf{R}$ that has a permutation of rows that results in columns with consecutive ones is said to have the *Consecutive One's Property* (C1P). The problem has been extensively studied and has many applications [4, 8, 9, 11]. There are linear algorithms to find a permutation if it exists; the first, due to Booth and Lucker, was based on PQ-trees [4]. There are approximate algorithms when an exact solution does not exist [7]. Our speech application that follows admits C1P. As such, the interval representation of r results in a significant complexity reduction over the point representation.

Label-Reachability Filter with Label Pushing A modification of the label-reachability filter for the case of a single transition matching leads to smaller and more efficient compositions as we will show in Section 3.

When matching an ϵ-transition e_1 in q_1 with an ϵ^L-loop in q_2, the Φ_{reach} filter allows this match if and only the set of transitions in q_2 that match the future in $n[e_1]$ is non-empty. In the special case where this set contains a unique transition e'_2, the $\Phi_{\text{push-label}}$ filter allows e_1 to match e'_2, resulting in the early output of $o[e'_2]$.

For $\Phi_{\text{push-label}}$, let $Q_3 = \{\epsilon, \perp\} \cup \mathcal{B}$, $i_3 = \epsilon$ and $\rho(q_3) = \overline{1}$ if $q_3 = \epsilon$ and $\rho(q_3) = \overline{0}$ otherwise. Let $\varphi(e_1, e_2, q_3) = (e_1, e_2, \epsilon)$ if $q_3 = \epsilon$ and $o[e_1] = i[e_2]$, or if $q_3 = o[e_1] = \epsilon$, $i[e_2] = \epsilon^L$ and $|\{e \in E[q_2] : r(n[e_1], i[e]) = 1\}| \geq 2$, or if $q_3 = o[e_1] \neq \epsilon$ and $i[e_2] = \epsilon^L$. Let $\varphi(e_1, e_2, q_3) = (e_1, e_2, q_3)$ if $q_3 \neq \epsilon$, $o[e_1] = \epsilon$, $i[e_2] =$

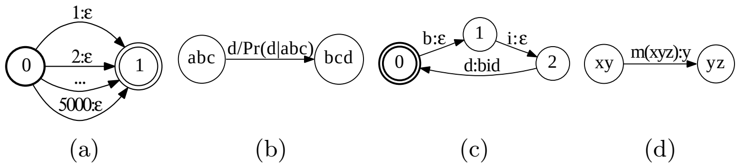

Fig. 2. Example transducers: (a) deleting transducer D, (b) n-gram language model G transition, (c) pronunciation lexicon L path, and (d) context-dependency transducer C transition.

ϵ^L and $r(n[e_1], q_3) = 1$. Let $\varphi(e_1, e_2, \epsilon) = (e_1, e'_2, i[e'_2])$ if $o[e_1] = \epsilon$, $i[e_2] = \epsilon^L$ and $\{e \in E[q_2] : r(n[e_1], i[e]) = 1\} = \{e'_2\}$. Otherwise, let $\varphi(e_1, e_2, q_3) = (e_1, e_2, \bot)$.

The complexity of the algorithm is the same as when using the label-reachability filter.

2.5 Combining filters

In Section 2.3 we presented composition filters for correctly handling epsilon transitions and in Section 2.4 we presented look-ahead filters that can lead to more efficient composition. In practice, we may need a combination of these filters, for example, to match with epsilon transitions and look-ahead along paths in a particular way. We present here how to synthesize a new composition filter from two components filters.

Let $\Phi^a = (Q_3^a, i_3^a, \bot^a, \varphi^a, \rho_3^a)$ and $\Phi^b = (Q_3^b, i_3^b, \bot^b, \varphi^b, \rho_3^b)$ be two composition filters, we will define their combination as the filter $\Phi^a \diamond \Phi^b = (Q_3, i_3, \bot, \varphi, \rho_3)$ with $Q_3 = Q_3^a \times Q_3^b$, $i_3 = (i_3^a, i_3^b)$, $\bot = (\bot^a, \bot^b)$, $\rho_3((q_3^a, q_3^b)) = \rho_3^a(q_3^a) \otimes \rho_3^b(q_3^b)$, and with φ defined as follows: given $(e_1, e_2, q_3) \in E_1 \times E_2 \times Q_3$ with $q_3 = (q_3^a, q_3^b)$, $\varphi^b(e_1, e_2, q_3^b) = (e'_1, e'_2, r_3^b)$ and $\varphi^a(e'_1, e'_2, q_3^a) = (e''_1, e''_2, r_3^a)$, then let

$$\varphi(e_1, e_2, q_3) = (e''_1, e''_2, q'_3) \text{ with } q'_3 = \begin{cases} \bot & \text{if } r_3^a = \bot^a \text{ or } r_3^b = \bot^b, \\ (r_3^a, r_3^b) & \text{otherwise.} \end{cases}$$

The filter $\Phi_{\text{reach}} \diamond \overline{\Phi}_{\epsilon\text{-seq}}$ can for instance be used to benefit from the label-reachable filter when T_2 contains input ϵ-transitions.

3 Examples

In this section, examples are given of the previously-defined composition filters. All examples are benchmarked using the composition algorithm in *OpenFst* [3].

Let $\Sigma = \{1, \ldots, 5000\}$ and let D be the two-state transducer over $\Sigma \times \Sigma$ that transduces each input symbol to ϵ as depicted in Figure 2(a). Consider the composition $D \circ D^{-1}$ using the epsilon-matching and epsilon-sequencing filters. The former creates a two-state machine with a transition for every element of $\Sigma \times \Sigma$ while the latter is identical to the concatenation TT^{-1}. Table 1(a)-(b) compares the number of composition states, transitions, time and memory usage with these two filters. In this example, the epsilon-sequencing filter gives a much smaller and efficiently-generated result than the epsilon-matching filter. It is easy to find examples where the opposite is true.

For the look-ahead filters, we draw our examples from a standard large-vocabulary speech recognition task - DARPA Broadcast News (BN). There are three alphabets for this task: Ω, the set of BN English words used where $|\Omega| =$ 70,897; Π, the set of English phonemes where $|\Pi| = 46$; and Υ, a set of English tri-phonemic acoustic models where $|\Upsilon| =$ 20,910. There are three component transducers for this task:

- a 4-gram *language model* G, which is a weighted automaton over Ω and has 2,213,539 states and 10,225,015 transitions. The weights model the probability of a particular sentence being uttered as estimated from the BN corpus. Figure 2(b) depicts the 4-gram transition *abcd* in G with probablity $Pr(d|abc)$.
- a minimal deterministic *lexicon transducer* L over $\Omega \times \Pi$, which maps phonemic pronunications to their word symbols and has 63,283 states and 145,710 transitions. The pronunciations are from a pronunciation dictionary. Figure 2(c) depicts a path in L.
- a minimal deterministic tri-phonemic *context-dependency transducer* C over $\Upsilon \times \Pi$, which maps from tri-phonemic model sequences to their corresponding phonemic sequence and has 1454 states and 88,840 transitions. The acoustic models are produced in the acoustic training phase of speech recognition and model a phoneme in its left and right context (possibly clustered due to data sparsity). Figure 2(d) depicts the transition in C for the triphonemic *xyz* model, *m(xyz)*.

For precise details about their form and construction of these three transducers, see [13]. We have chosen these transducers since the composition $C \circ L \circ G$, mapping from tri-phonemic models to word sequences weighted by their probabilities, is the *recognition transducer* matched against acoustic input during the recognition of an utterance. However, both C and L present significant issues for classical composition as detailed below. By constructing C and L differently, it is possible to use classical composition more efficiently, however these constructions introduce considerable non-determinism in the result that requires an expensive determinization to remove, something that we often wish to avoid.

While these examples are drawn from speech recognition, other application areas (e.g. text-to-speech synthesis, optical character recognition, spelling correction) involve similar language models, dictionaries and/or context-dependent constraints that can be modeled usefully with transducers and present similar issues with composition.

In the examples below that involve ϵ-transitions, we in fact use look-ahead filters combined with the epsilon-sequencing filter as described in Section 2.5.

String-Potential Filter: As depicted in Figure 2(d), a single symbol (the right tri-phoneme) is the output label for each transition leaving a state in the C transducer. That symbol is also the string potential at each state. In composition, we can take advantage of this as demonstrated by Table 1(c)-(d), which compares C composed with a random string $\alpha \in \Pi^{1000000}$ using the trivial versus the

Table 1. Number of composition states and transitions (before trimming), time and memory usage for various composition filters. Observe that (a), (c), (e) and (g) correspond to using the composition algorithm from [13]. Experiments were conducted on a quad-core 2.2 GHz AMD Opteron machine with 32 GB of RAM.

	composition filter	$\mathbf{T_1}$	$\mathbf{T_2}$	$\mathbf{T_1 \circ T_2}$ **states**	$\mathbf{T_1 \circ T_2}$ **transitions**	**time (sec)**	**mem. (mbytes)**
(a)	`epsilon-matching`	D	D^{-1}	2	25,000,000	4.21	1419.5
(b)	`epsilon-sequencing`	D	D^{-1}	3	10,000	0.73	22.0
(c)	`trivial`	C	α	47,021,923	47,021,922	48.45	4704.0
(d)	`string-potential`	C	α	1,043,734	1,043,733	8.97	351.0
(e)	`trivial`	C	L	1,952,555	3,527,612	2.77	225.0
(f)	`transition-look-ahead`	C	L	120,489	149,972	0.84	33.4
(g)	`epsilon-sequencing`	L	G	?	?	$>$ 7200.00	$>$ 32,768.0
(h)	`label-reachability`	L	G	30,884,222	39,965,633	177.93	3612.9
(i)	`lab.-reach. w/ label-pushing`	L	G	13,377,323	22,151,870	113.72	1885.9

string-potential filters. The trivial filter is inefficient due to the output nondeterminism, while the string-potential filter is much better in both time and space. Another effective use of string potentials in composition is given in [2].

Transition-Look-Ahead Filter: Unlike the previous example, the composition $C \circ L$ will not benefit much from using the string-potential filter since the string potential at most states in L is ϵ. In this case, the transition-look-ahead filter can be applied. Table 1(e)-(f), which compares the trivial and transition-look-ahead filters, demonstrates that the transition-look-ahead filter creates fewer states in the (untrimmed) result, saving time and space.

Label-Reachability Filter: The composition $L \circ G$ using the epsilon-sequencing (or -matching) composition filter is very inefficient since the initial epsilon paths in L create many non-coaccessible states in the result. For this problem, the label-reachability filter is appropriate. Table 1(g)-(h) compares the epsilon-sequencing and label-reachability filters. With the epsilon-sequencing filter, composition terminates after 2 hours with RAM exhausted, while with the label-reachability filter, only a few minutes are needed for completion.

Label-Reachability Filter with Label Pushing: While the label-reachability filter addresses the non-coaccessible states in the composition $L \circ G$ (in fact, the result is trim), it can further benefit from including label-pushing in the filter. Table 1(i) shows that if we do so, the result is smaller, builds faster and uses less memory. This benefit is due, in part, to all transitions entering a state in G having the same label.

4 Implementation

In *OpenFst* [3], the default composition filter is the epsilon-sequencing filter. It can be easily and very efficiently changed via templated options. For example, to use the epsilon-matching filter, one invokes:

```
ComposeFstOptions<StdArc, MatchComposeFilter> opts;
ComposeFst<StdArc> result(t1, t2, opts);
```

All filters described here are available in *OpenFst*. Further, users can add new ones by creating a class that meets the composition filter interface to handle their specific applications.

Acknowledgements We thank Mehryar Mohri for suggesting using a generalized composition filter for solving problems such as those addressed here.

References

1. C. Allauzen, M. Mohri, and A. Rastogi. General algorithms for testing the ambiguity of finite automata. In *DLT*, volume 5257 of *LNCS*, pages 108–120, 2008.
2. C. Allauzen, M. Mohri, and M. Riley. Statistical modeling for unit selection in speech synthesis. In *Proc. ACL*, pages 55–62, 2004.
3. C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri. OpenFst: A general and efficient weighted finite-state transducer library. In *CIAA*, volume 4783 of *LNCS*, pages 11–23, 2007. `http://www.openfst.org`.
4. K. Booth and G. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *J. of Computer and System Sci.*, 13:335–379, 1976.
5. D. Caseiro and I. Trancoso. A specialized on-the-fly algorithm for lexicon and language model composition. *IEEE Trans. on Audio, Speech and Lang. Proc.*, 14(4):1281–1291, 2006.
6. O. Cheng, J. Dines, and M. Doss. A generalized dynamic composition algorithm of weighted finite state transducers for large vocabulary speech recognition. In *Proc. ICASSP*, volume 4, pages 345–348, 2007.
7. M. Dom and R. Niedermeier. The search for consecutive ones submatrices: Faster and more general. In *Proc. ACID*, pages 43–54, 2007.
8. M. Habib, R. McConnell, C. Paul, and L. Viennot. Lex-BFS and partition refinement with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theor. Comput. Sci.*, 234:59–84, 2000.
9. W.-L. Hsu and R. McConnell. PC trees and circular-ones arrangements. *Theor. Comput. Sci.*, 296(1):99–116, 2003.
10. J. McDonough, E. Stoimenov, and D. Klakow. An algorithm for fast composition of weighted finite-state transducers. In *Proc. ASRU*, 2007.
11. J. Meidanis, O. Porto, and G. Telles. On the consecutive ones property. *Discrete Appl. Math.*, 88:325–354, 1998.
12. M. Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.
13. M. Mohri, F. Pereira, and M. Riley. Speech recognition with weighted finite-state transducers. In Y. H. Jacob Benesty, Mohan Sondhi, editor, *Handbook of Speech Processing*, pages 559–582. Springer, 2008.
14. T. Oonishi, P. Dixon, K. Iwano, and S. Furui. Implementation and evaluation of fast on-the-fly WFST composition algorithms. In *Proc. Interspeech*, pages 2110–2113, 2008.