Efficient Inference Through Cascades of Weighted Tree Transducers

Jonathan May and Kevin Knight
Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292
{jonmay,knight}@isi.edu

Heiko Vogler
Technische Universität Dresden
Institut für Theoretische Informatik
01062 Dresden, Germany
heiko.vogler@tu-dresden.de

Abstract

Weighted tree transducers have been proposed as useful formal models for representing syntactic natural language processing applications, but there has been little description of inference algorithms for these automata beyond formal foundations. We give a detailed description of algorithms for application of cascades of weighted tree transducers to weighted tree acceptors, connecting formal theory with actual practice. Additionally, we present novel on-the-fly variants of these algorithms, and compare their performance on a syntax machine translation cascade based on (Yamada and Knight, 2001).

1 Motivation

Weighted finite-state transducers have found recent favor as models of natural language (Mohri, 1997). In order to make actual use of systems built with these formalisms we must first calculate the set of possible weighted outputs allowed by the transducer given some input, which we call *forward application*, or the set of possible weighted inputs given some output, which we call *backward application*. After application we can do some inference on this result, such as determining its k highest weighted elements.

We may also want to divide up our problems into manageable chunks, each represented by a transducer. As noted by Woods (1980), it is easier for designers to write several small transducers where each performs a simple transformation, rather than painstakingly construct a single complicated device. We would like to know, then, the result of transformation of input or output by a *cascade* of transducers, one operating after the other. As we will see, there are various strategies for approaching this problem. We will consider *offline composition*, *bucket brigade application*, and *on-the-fly application*.

Application of cascades of weighted string transducers (WSTs) has been well-studied (Mohri, 1997). Less well-studied but of more recent interest is application of cascades of weighted *tree* transducers (WTTs). We tackle application of WTT cascades in this work, presenting:

- explicit algorithms for application of WTT cascades
- novel algorithms for on-the-fly application of WTT cascades, and
- experiments comparing the performance of these algorithms.

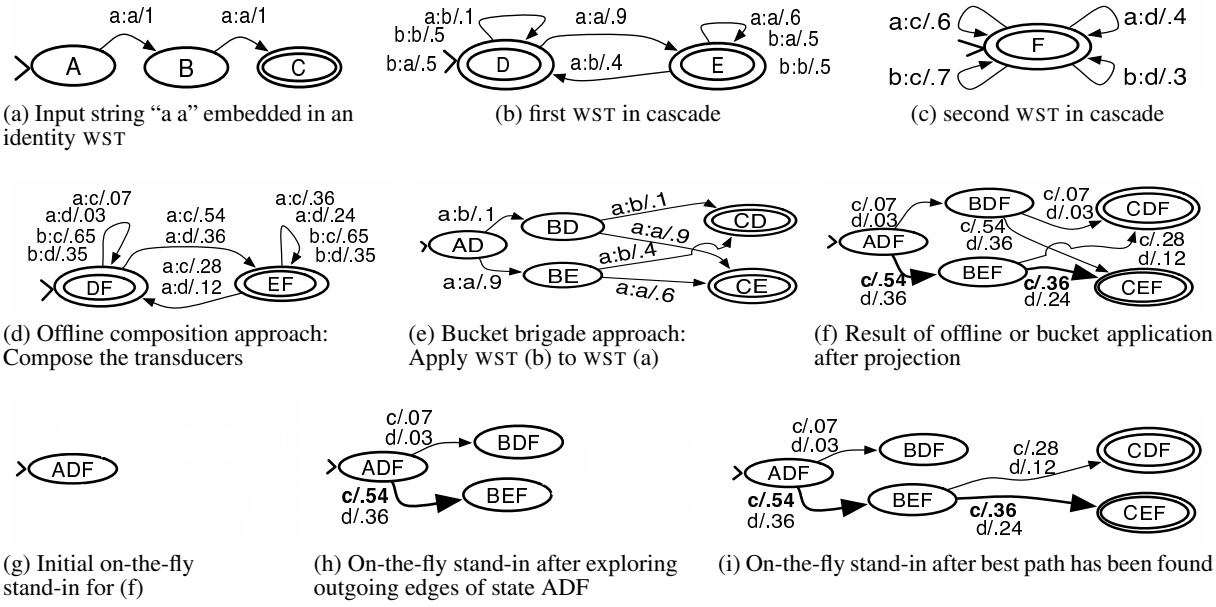
2 Strategies for the string case

Before we discuss application of WTTs, it is helpful to recall the solution to this problem in the WST domain. We recall previous formal presentations of WSTs (Mohri, 1997) and note informally that they may be represented as directed graphs with designated start and end states and edges labeled with input symbols, output symbols, and weights.[1] Fortunately, the solution for WSTs is practically trivial—we achieve application through a series of *embedding*, *composition*, and *projection* operations. Embedding is simply the act of representing a string or regular string language as an identity WST. Composition of WSTs, that is, generating a single WST that captures the transformations of two input WSTs used in sequence, is not at all trivial, but has been well covered in, e.g., (Mohri, 2009), where directly implementable algorithms can be found. Finally, projection is another trivial operation—the domain or range language can be obtained from a WST by ignoring the output or input symbols, respectively, on its arcs, and summing weights on otherwise identical arcs. By embedding an input, composing the result with the given WST, and projecting the result, forward application is accomplished.[2] We are then left with a weighted string acceptor (WSA), essentially a weighted, labeled graph, which can be traversed

[1] We assume throughout this paper that weights are in $\mathbb{R}_+ \cup \{+\infty\}$, that the weight of a path is calculated as the product of the weights of its edges, and that the weight of a (not necessarily finite) set T of paths is calculated as the sum of the weights of the paths of T.

[2] For backward applications, the roles of input and output are simply exchanged.

Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, pages 1058–1066,
Uppsala, Sweden, 11-16 July 2010.



(a) Input string "a a" embedded in an identity WST

(b) first WST in cascade

(c) second WST in cascade

(d) Offline composition approach: Compose the transducers

(e) Bucket brigade approach: Apply WST (b) to WST (a)

(f) Result of offline or bucket application after projection

(g) Initial on-the-fly stand-in for (f)

(h) On-the-fly stand-in after exploring outgoing edges of state ADF

(i) On-the-fly stand-in after best path has been found

Figure 1: Three different approaches to application through cascades of WSTs.

by well-known algorithms to efficiently find the k-best paths.

Because WSTs can be freely composed, extending application to operate on a *cascade* of WSTs is fairly trivial. The only question is one of composition order: whether to initially compose the cascade into a single transducer (an approach we call *offline composition*) or to compose the initial embedding with the first transducer, trim useless states, compose the result with the second, and so on (an approach we call *bucket brigade*). The appropriate strategy generally depends on the structure of the individual transducers.

A third approach builds the result incrementally, as dictated by some algorithm that requests information about it. Such an approach, which we call *on-the-fly*, was described in (Pereira and Riley, 1997; Mohri, 2009; Mohri et al., 2000). If we can efficiently calculate the outgoing edges of a state of the result WSA on demand, without calculating all edges in the entire machine, we can maintain a *stand-in* for the result structure, a machine consisting at first of only the start state of the true result. As a calling algorithm (e.g., an implementation of Dijkstra's algorithm) requests information about the result graph, such as the set of outgoing edges from a state, we replace the current stand-in with a richer version by adding the result of the request. The on-the-fly approach has a distinct advantage over the other two methods in that the entire result graph need not be built. A graphical representation of all three methods is presented in Figure 1.

3 Application of tree transducers

Now let us revisit these strategies in the setting of trees and tree transducers. Imagine we have a tree or set of trees as input that can be represented as a weighted regular tree grammar[3] (WRTG) and a WTT that can transform that input with some weight. We would like to know the k-best trees the WTT can produce as output for that input, along with their weights. We already know of several methods for acquiring k-best trees from a WRTG (Huang and Chiang, 2005; Pauls and Klein, 2009), so we then must ask if, analogously to the string case, WTTs preserve recognizability[4] and we can form an application WRTG. Before we begin, however, we must define WTTs and WRTGs.

3.1 Preliminaries[5]

A *ranked alphabet* is a finite set Σ such that every member $\sigma \in \Sigma$ has a rank $rk(\sigma) \in \mathbb{N}$. We call $\Sigma^{(k)} \subseteq \Sigma, k \in \mathbb{N}$ the set of those $\sigma \in \Sigma$ such that $rk(\sigma) = k$. The set of *variables* is denoted $X = \{x_1, x_2, \ldots\}$ and is assumed to be disjoint from any ranked alphabet used in this paper. We use $\perp$ to denote a symbol of rank 0 that is not in any ranked alphabet used in this paper. A *tree* $t \in T_\Sigma$ is denoted $\sigma(t_1, \ldots, t_k)$ where $k \geq 0$, $\sigma \in \Sigma^{(k)}$, and $t_1, \ldots, t_k \in T_\Sigma$. For $\sigma \in \Sigma^{(0)}$ we

[3] This generates the same class of weighted tree languages as weighted tree automata, the direct analogue of WSAs, and is more useful for our purposes.

[4] A weighted tree language is recognizable iff it can be represented by a wrtg.

[5] The following formal definitions and notations are needed for understanding and reimplementation of the presented algorithms, but can be safely skipped on first reading and consulted when encountering an unfamiliar term.

write $\sigma \in T_\Sigma$ as shorthand for $\sigma()$. For every set S disjoint from Σ, let $T_\Sigma(S) = T_{\Sigma \cup S}$, where, for all $s \in S$, $rk(s) = 0$.

We define the *positions* of a tree $t = \sigma(t_1, \ldots, t_k)$, for $k \geq 0$, $\sigma \in \Sigma^{(k)}$, $t_1, \ldots, t_k \in T_\Sigma$, as a set $pos(t) \subset \mathbb{N}^*$ such that $pos(t) = \{\varepsilon\} \cup \{iv \mid 1 \leq i \leq k, v \in pos(t_i)\}$. The set of leaf positions $lv(t) \subseteq pos(t)$ are those positions $v \in pos(t)$ such that for no $i \in \mathbb{N}$, $vi \in pos(t)$. We presume standard lexicographic orderings $<$ and $\leq$ on pos.

Let $t, s \in T_\Sigma$ and $v \in pos(t)$. The *label* of t at position v, denoted by $t(v)$, the *subtree* of t at v, denoted by $t|_v$, and the *replacement* at v by s, denoted by $t[s]_v$, are defined as follows:

1. For every $\sigma \in \Sigma^{(0)}$, $\sigma(\varepsilon) = \sigma$, $\sigma|_\varepsilon = \sigma$, and $\sigma[s]_\varepsilon = s$.
2. For every $t = \sigma(t_1, \ldots, t_k)$ such that $k = rk(\sigma)$ and $k \geq 1$, $t(\varepsilon) = \sigma$, $t|_\varepsilon = t$, and $t[s]_\varepsilon = s$. For every $1 \leq i \leq k$ and $v \in pos(t_i)$, $t(iv) = t_i(v)$, $t|_{iv} = t_i|_v$, and $t[s]_{iv} = \sigma(t_1, \ldots, t_{i-1}, t_i[s]_v, t_{i+1}, \ldots, t_k)$.

The *size* of a tree t, $size(t)$ is $|pos(t)|$, the cardinality of its position set. The *yield set* of a tree is the set of labels of its leaves: for a tree t, $yd(t) = \{t(v) \mid v \in lv(t)\}$.

Let A and B be sets. Let $\varphi : A \to T_\Sigma(B)$ be a mapping. We extend φ to the mapping $\overline{\varphi} : T_\Sigma(A) \to T_\Sigma(B)$ such that for $a \in A$, $\overline{\varphi}(a) = \varphi(a)$ and for $k \geq 0$, $\sigma \in \Sigma^{(k)}$, and $t_1, \ldots, t_k \in T_\Sigma(A)$, $\overline{\varphi}(\sigma(t_1, \ldots, t_k)) = \sigma(\overline{\varphi}(t_1), \ldots, \overline{\varphi}(t_k))$. We indicate such extensions by describing φ as a *substitution mapping* and then using $\overline{\varphi}$ without further comment.

We use $\mathbb{R}_+$ to denote the set $\{w \in \mathbb{R} \mid w \geq 0\}$ and $\mathbb{R}_+^\infty$ to denote $\mathbb{R}_+ \cup \{+\infty\}$.

Definition 3.1 (cf. (Alexandrakis and Bozapalidis, 1987)) A weighted regular tree grammar (WRTG) is a 4-tuple $G = (N, \Sigma, P, n_0)$ where:

1. N is a finite set of *nonterminals*, with $n_0 \in N$ the start nonterminal.
2. Σ is a ranked alphabet of input symbols, where $\Sigma \cap N = \emptyset$.
3. P is a tuple (P', π), where P' is a finite set of *productions*, each production p of the form $n \to u$, $n \in N$, $u \in T_\Sigma(N)$, and $\pi : P' \to \mathbb{R}_+$ is a weight function of the productions. We will refer to P as a finite set of weighted productions, each production p of the form $n \xrightarrow{\pi(p)} u$.

A production p is a *chain production* if it is of the form $n_i \xrightarrow{w} n_j$, where $n_i, n_j \in N$.[6]

[6] In (Alexandrakis and Bozapalidis, 1987), chain productions are forbidden in order to avoid infinite summations. We explicitly allow such summations.

A WRTG G is in *normal form* if each production is either a chain production or is of the form $n \xrightarrow{w} \sigma(n_1, \ldots, n_k)$ where $\sigma \in \Sigma^{(k)}$ and $n_1, \ldots, n_k \in N$.

For WRTG $G = (N, \Sigma, P, n_0)$, $s, t, u \in T_\Sigma(N)$, $n \in N$, and $p \in P$ of the form $n \xrightarrow{w} u$, we obtain a *derivation step* from s to t by replacing some leaf nonterminal in s labeled n with u. Formally, $s \Rightarrow_G^p t$ if there exists some $v \in lv(s)$ such that $s(v) = n$ and $s[u]_v = t$. We say this derivation step is *leftmost* if, for all $v' \in lv(s)$ where $v' < v$, $s(v') \in \Sigma$. We henceforth assume all derivation steps are leftmost. If, for some $m \in \mathbb{N}$, $p_i \in P$, and $t_i \in T_\Sigma(N)$ for all $1 \leq i \leq m$, $n_0 \Rightarrow^{p_1} t_1 \ldots \Rightarrow^{p_m} t_m$, we say the sequence $d = (p_1, \ldots, p_m)$ is a *derivation* of t_m in G and that $n_0 \Rightarrow^* t_m$; the weight of d is $wt(d) = \pi(p_1) \cdot \ldots \cdot \pi(p_m)$. The weighted tree language recognized by G is the mapping $L_G : T_\Sigma \to \mathbb{R}_+^\infty$ such that for every $t \in T_\Sigma$, $L_G(t)$ is the sum of the weights of all (possibly infinitely many) derivations of t in G. A weighted tree language $f : T_\Sigma \to \mathbb{R}_+^\infty$ is *recognizable* if there is a WRTG G such that $f = L_G$.

We define a partial ordering $\preceq$ on WRTGs such that for WRTGs $G_1 = (N_1, \Sigma, P_1, n_0)$ and $G_2 = (N_2, \Sigma, P_2, n_0)$, we say $G_1 \preceq G_2$ iff $N_1 \subseteq N_2$ and $P_1 \subseteq P_2$, where the weights are preserved.

Definition 3.2 (cf. Def. 1 of (Maletti, 2008)) A weighted extended top-down tree transducer (WXTT) is a 5-tuple $M = (Q, \Sigma, \Delta, R, q_0)$ where:

1. Q is a finite set of states.
2. Σ and Δ are the ranked alphabets of input and output symbols, respectively, where $(\Sigma \cup \Delta) \cap Q = \emptyset$.
3. R is a tuple (R', π), where R' is a finite set of *rules*, each rule r of the form $q.y \to u$ for $q \in Q$, $y \in T_\Sigma(X)$, and $u \in T_\Delta(Q \times X)$. We further require that no variable $x \in X$ appears more than once in y, and that each variable appearing in u is also in y. Moreover, $\pi : R' \to \mathbb{R}_+^\infty$ is a weight function of the rules. As for WRTGs, we refer to R as a finite set of weighted rules, each rule r of the form $q.y \xrightarrow{\pi(r)} u$.

A WXTT is *linear* (respectively, *nondeleting*) if, for each rule r of the form $q.y \xrightarrow{w} u$, each $x \in yd(y) \cap X$ appears at most once (respectively, at least once) in u. We denote the class of all WXTTs as wxT and add the letters L and N to signify the subclasses of linear and nondeleting WTT, respectively. Additionally, if y is of the form $\sigma(x_1, \ldots, x_k)$, we remove the letter "x" to signify

the transducer is not extended (i.e., it is a "traditional" WTT (Fülöp and Vogler, 2009)).

For WXTT $M = (Q, \Sigma, \Delta, R, q_0)$, $s, t \in T_\Delta(Q \times T_\Sigma)$, and $r \in R$ of the form $q.y \xrightarrow{w} u$, we obtain a *derivation step* from s to t by replacing some leaf of s labeled with q and a tree matching y by a transformation of u, where each instance of a variable has been replaced by a corresponding subtree of the y-matching tree. Formally, $s \Rightarrow_M^r t$ if there is a position $v \in pos(s)$, a substitution mapping $\varphi : X \to T_\Sigma$, and a rule $q.y \xrightarrow{w} u \in R$ such that $s(v) = (q, \overline{\varphi}(y))$ and $t = s[\overline{\varphi'}(u)]_v$, where φ' is a substitution mapping $Q \times X \to T_\Delta(Q \times T_\Sigma)$ defined such that $\varphi'(q', x) = (q', \varphi(x))$ for all $q' \in Q$ and $x \in X$. We say this derivation step is *leftmost* if, for all $v' \in lv(s)$ where $v' < v$, $s(v') \in \Delta$. We henceforth assume all derivation steps are leftmost. If, for some $s \in T_\Sigma$, $m \in \mathbb{N}$, $r_i \in R$, and $t_i \in T_\Delta(Q \times T_\Sigma)$ for all $1 \leq i \leq m$, $(q_0, s) \Rightarrow^{r_1} t_1 \ldots \Rightarrow^{r_m} t_m$, we say the sequence $d = (r_1, \ldots, r_m)$ is a *derivation* of (s, t_m) in M; the weight of d is $wt(d) = \pi(r_1) \cdot \ldots \cdot \pi(r_m)$. The *weighted tree transformation* recognized by M is the mapping $\tau_M : T_\Sigma \times T_\Delta \to \mathbb{R}_+^\infty$, such that for every $s \in T_\Sigma$ and $t \in T_\Delta$, $\tau_M(s, t)$ is the sum of the weights of all (possibly infinitely many) derivations of (s, t) in M. The *composition* of two weighted tree transformations $\tau : T_\Sigma \times T_\Delta \to \mathbb{R}_+^\infty$ and $\mu : T_\Delta \times T_\Gamma \to \mathbb{R}_+^\infty$ is the weighted tree transformation $(\tau; \mu) : T_\Sigma \times T_\Gamma \to \mathbb{R}_+^\infty$ where for every $s \in T_\Sigma$ and $u \in T_\Gamma$, $(\tau; \mu)(s, u) = \sum_{t \in T_\Delta} \tau(s, t) \cdot \mu(t, u)$.

3.2 Applicable classes

We now consider transducer classes where recognizability is preserved under application. Table 1 presents known results for the top-down tree transducer classes described in Section 3.1. Unlike the string case, preservation of recognizability is not universal or symmetric. This is important for us, because we can only construct an *application* WRTG, i.e., a WRTG representing the result of application, if we can ensure that the language generated by application is in fact recognizable. Of the types under consideration, only wxLNT and wLNT preserve forward recognizability. The two classes marked as open questions and the other classes, which are superclasses of wNT, do not or are presumed not to. All subclasses of wxLT preserve backward recognizability.[7] We do not consider cases where recognizability is not preserved in the remainder of this paper. If a transducer M of a class that preserves forward recognizability is applied to a WRTG G, we can call the forward application WRTG $M(G)^\triangleright$ and if M preserves backward recognizability, we can call the backward application WRTG $M(G)^\triangleleft$.

Now that we have explained the application problem in the context of weighted tree transducers and determined the classes for which application is possible, let us consider how to build forward and backward application WRTGs. Our basic approach mimics that taken for WSTs by using an embed-compose-project strategy. As in string world, if we can embed the input in a transducer, compose with the given transducer, and project the result, we can obtain the application WRTG. Embedding a WRTG in a wLNT is a trivial operation—if the WRTG is in normal form and chain production-free,[8] for every production of the form $n \xrightarrow{w} \sigma(n_1, \ldots, n_k)$, create a rule of the form $n.\sigma(x_1, \ldots, x_k) \xrightarrow{w} \sigma(n_1.x_1, \ldots, n_k.x_k)$. Range projection of a wxLNT is also trivial—for every $q \in Q$ and $u \in T_\Delta(Q \times X)$ create a production of the form $q \xrightarrow{w} u'$ where u' is formed from u by replacing all leaves of the form $q.x$ with the leaf q, i.e., removing references to variables, and w is the sum of the weights of all rules of the form $q.y \to u$ in R.[9] Domain projection for wxLT is best explained by way of example. The left side of a rule is preserved, with variables leaves replaced by their associated states from the right side. So, the rule $q_1.\sigma(\gamma(x_1), x_2) \xrightarrow{w} \delta(q_2.x_2, \beta(\alpha, q_3.x_1))$ would yield the production $q_1 \xrightarrow{w} \sigma(\gamma(q_3), q_2)$ in the domain projection. However, a deleting rule such as $q_1.\sigma(x_1, x_2) \xrightarrow{w} \gamma(q_2.x_2)$ necessitates the introduction of a new nonterminal $\perp$ that can generate all of T_Σ with weight 1.

The only missing piece in our embed-compose-project strategy is composition. Algorithm 1, which is based on the declarative construction of Maletti (2006), generates the syntactic composition of a wxLT and a wLNT, a generalization of the basic composition construction of Baker (1979). It calls Algorithm 2, which determines the sequences of rules in the second transducer that match the right side of a single rule in the first transducer. Since the embedded WRTG is of type wLNT, it may be either the first or second argument provided to Algorithm 1, depending on whether the application is forward or backward. We can thus use the embed-compose-project strategy for forward application of wLNT and backward application of wxLT and wxLNT. Note that we *cannot* use this strategy for forward applica-

[7] Note that the introduction of weights limits recognizability preservation considerably. For example, (unweighted) xT preserves backward recognizability.

[8] Without loss of generality we assume this is so, since standard algorithms exist to remove chain productions (Kuich, 1998; Ésik and Kuich, 2003; Mohri, 2009) and convert into normal form (Alexandrakis and Bozapalidis, 1987).

[9] Finitely many such productions may be formed.

tion of wxLNT, even though that class preserves recognizability.

Algorithm 1 COMPOSE

inputs
  wxLT $M_1 = (Q_1, \Sigma, \Delta, R_1, q_{1_0})$
  wLNT $M_2 = (Q_2, \Delta, \Gamma, R_2, q_{2_0})$
outputs
  wxLT $M_3 = ((Q_1 \times Q_2), \Sigma, \Gamma, R_3, (q_{1_0}, q_{2_0}))$ such that $M_3 = (\tau_{M_1}; \tau_{M_2})$.
complexity
  $O(|R_1| \max(|R_2|^{size(\tilde{u})}, |Q_2|))$, where $\tilde{u}$ is the largest right side tree in any rule in R_1

Let R_3 be of the form (R'_3, π)
$R_3 \leftarrow (\emptyset, \emptyset)$
$\Xi \leftarrow \{(q_{1_0}, q_{2_0})\}$ {seen states}
$\Psi \leftarrow \{(q_{1_0}, q_{2_0})\}$ {pending states}
while $\Psi \neq \emptyset$ **do**
  $(q_1, q_2) \leftarrow$any element of Ψ
  $\Psi \leftarrow \Psi \setminus \{(q_1, q_2)\}$
  for all $(q_1.y \xrightarrow{w_1} u) \in R_1$ **do**
    for all $(z, w_2) \in \text{COVER}(u, M_2, q_2)$ **do**
      for all $(q, x) \in yd(z) \cap ((Q_1 \times Q_2) \times X)$ **do**
        if $q \notin \Xi$ **then**
          $\Xi \leftarrow \Xi \cup \{q\}$
          $\Psi \leftarrow \Psi \cup \{q\}$
        $r \leftarrow ((q_1, q_2).y \rightarrow z)$
        $R'_3 \leftarrow R'_3 \cup \{r\}$
        $\pi(r) \leftarrow \pi(r) + (w_1 \cdot w_2)$
return M_3

4 Application of tree transducer cascades

What about the case of an input WRTG and a cascade of tree transducers? We will revisit the three strategies for accomplishing application discussed above for the string case.

In order for offline composition to be a viable strategy, the transducers in the cascade must be closed under composition. Unfortunately, of the classes that preserve recognizability, only wLNT is closed under composition (Gécseg and Steinby, 1984; Baker, 1979; Maletti et al., 2009; Fülöp and Vogler, 2009).

However, the general lack of composability of tree transducers does *not* preclude us from conducting forward application of a cascade. We revisit the bucket brigade approach, which in Section 2 appeared to be little more than a choice of composition order. As discussed previously, application of a single transducer involves an embedding, a composition, and a projection. The embedded WRTG is in the class wLNT, and the projection forms another WRTG. As long as every transducer in the cascade can be composed with a wLNT to its left or right, depending on the application type, application of a cascade is possible. Note that this embed-compose-project process is somewhat more burdensome than in the string case. For strings, application is obtained by a *single* embedding, a series of compositions, and a *single* projection, whereas application for trees is obtained by a series of (embed, compose, project) operations.

Algorithm 2 COVER

inputs
  $u \in T_\Delta(Q_1 \times X)$
  wT $M_2 = (Q_2, \Delta, \Gamma, R_2, q_{2_0})$
  state $q_2 \in Q_2$
outputs
  set of pairs (z, w) with $z \in T_\Gamma((Q_1 \times Q_2) \times X)$ formed by one or more successful runs on u by rules in R_2, starting from q_2, and $w \in \mathbb{R}^\infty_+$ the sum of the weights of all such runs.
complexity
  $O(|R_2|^{size(u)})$

if $u(\varepsilon)$ is of the form $(q_1, x) \in Q_1 \times X$ **then**
  $z_{init} \leftarrow ((q_1, q_2), x)$
else
  $z_{init} \leftarrow \perp$
$\Pi_{last} \leftarrow \{(z_{init}, \{((\varepsilon, \varepsilon), q_2)\}, 1)\}$
for all $v \in pos(u)$ such that $u(v) \in \Delta^{(k)}$ for some $k \geq 0$ in prefix order **do**
  $\Pi_v \leftarrow \emptyset$
  for all $(z, \theta, w) \in \Pi_{last}$ **do**
    for all $v' \in lv(z)$ such that $z(v') = \perp$ **do**
      for all $(\theta(v, v').u(v)(x_1, \ldots, x_k) \xrightarrow{w'} h) \in R_2$ **do**
        $\theta' \leftarrow \theta$
        Form substitution mapping $\varphi : (Q_2 \times X) \rightarrow T_\Gamma((Q_1 \times Q_2 \times X) \cup \{\perp\})$.
        for $i = 1$ **to** k **do**
          for all $v'' \in pos(h)$ such that $h(v'') = (q'_2, x_i)$ for some $q'_2 \in Q_2$ **do**
            $\theta'(vi, v'v'') \leftarrow q'_2$
            if $u(vi)$ is of the form $(q_1, x) \in Q_1 \times X$ **then**
              $\varphi(q'_2, x_i) \leftarrow ((q_1, q'_2), x)$
            else
              $\varphi(q'_2, x_i) \leftarrow \perp$
        $\Pi_v \leftarrow \Pi_v \cup \{(z[\overline{\varphi}(h)]_{v'}, \theta', w \cdot w')\}$
  $\Pi_{last} \leftarrow \Pi_v$
$Z \leftarrow \{z \mid (z, \theta, w) \in \Pi_{last}\}$
return $\{(z, \sum_{(z,\theta,w) \in \Pi_{last}} w) \mid z \in Z\}$

4.1 On-the-fly algorithms

We next consider on-the-fly algorithms for application. Similar to the string case, an on-the-fly approach is driven by a calling algorithm that periodically needs to know the productions in a WRTG with a common left side nonterminal. The embed-compose-project approach produces an entire application WRTG before any inference algorithm is run. In order to admit an on-the-fly approach we describe algorithms that only generate those productions in a WRTG that have a given left nonterminal. In this section we extend Definition 3.1 as follows: a WRTG is a 6-tuple $G = (N, \Sigma, P, n_0, \overline{M}, \overline{G})$ where N, Σ, P, and n_0 are defined as in Definition 3.1, and either $\overline{M} = \overline{G} = \emptyset$,[10] or $\overline{M}$ is a wxLNT and $\overline{G}$ is a normal form, chain production-free WRTG such that

[10] In which case the definition is functionally unchanged from before.

type	preserved?	source
w[x]T	No	See w[x]NT
w[x]LT	OQ	(Maletti, 2009)
w[x]NT	No	(Gécseg and Steinby, 1984)
wxLNT	Yes	(Fülöp et al., 2010)
wLNT	Yes	(Kuich, 1999)

(a) Preservation of forward recognizability

type	preserved?	source
w[x]T	No	See w[x]NT
w[x]LT	Yes	(Fülöp et al., 2010)
w[x]NT	No	(Maletti, 2009)
w[x]LNT	Yes	See w[x]LT

(b) Preservation of backward recognizability

Table 1: Preservation of forward and backward recognizability for various classes of top-down tree transducers. Here and elsewhere, the following abbreviations apply: w = weighted, x = extended LHS, L = linear, N = nondeleting, OQ = open question. Square brackets include a superposition of classes. For example, w[x]T signifies both wxT and wT.

Algorithm 3 PRODUCE

inputs
  WRTG $G_{in} = (N_{in}, \Delta, P_{in}, n_0, \overline{M}, \overline{G})$ such that $\overline{M} = (Q, \Sigma, \Delta, R, q_0)$ is a wxLNT and $\overline{G} = (N, \Sigma, P, n_0', M', G')$ is a WRTG in normal form with no chain productions
  $n_{in} \in N_{in}$
outputs
  WRTG G_{out} = $(N_{out}, \Delta, P_{out}, n_0, \overline{M}, \overline{G})$, such that $G_{in} \preceq G_{out}$ and $(n_{in} \xrightarrow{w} u) \in P_{out} \Leftrightarrow (n_{in} \xrightarrow{w} u) \in \overline{M}(\overline{G})^{\triangleright}$
complexity
  $O(|R||P|^{size(\tilde{y})})$, where $\tilde{y}$ is the largest left side tree in any rule in R

if P_{in} contains productions of the form $n_{in} \xrightarrow{w} u$ **then**
  return G_{in}
$N_{out} \leftarrow N_{in}$
$P_{out} \leftarrow P_{in}$
Let n_{in} be of the form (n, q), where $n \in N$ and $q \in Q$.
for all $(q.y \xrightarrow{w_1} u) \in R$ **do**
  for all $(\theta, w_2) \in$ REPLACE$(y, \overline{G}, n)$ **do**
    Form substitution mapping $\varphi : Q \times X \to T_{\Delta}(N \times Q)$ such that, for all $v \in yd(y)$ and $q' \in Q$, if there exist $n' \in N$ and $x \in X$ such that $\theta(v) = n'$ and $y(v) = x$, then $\varphi(q', x) = (n', q')$.
    $p' \leftarrow ((n, q) \xrightarrow{w_1 \cdot w_2} \overline{\varphi}(u))$
    for all $p \in$ NORM(p', N_{out}) **do**
      Let p be of the form $n_0 \xrightarrow{w} \delta(n_1, \ldots, n_k)$ for $\delta \in \Delta^{(k)}$.
      $N_{out} \leftarrow N_{out} \cup \{n_0, \ldots, n_k\}$
      $P_{out} \leftarrow P_{out} \cup \{p\}$
return CHAIN-REM(G_{out})

$G \preceq \overline{M}(\overline{G})^{\triangleright}$. In the latter case, G is a stand-in for $\overline{M}(\overline{G})^{\triangleright}$, analogous to the stand-ins for WSAs and WSTs described in Section 2.

Algorithm 3, PRODUCE, takes as input a WRTG $G_{in} = (N_{in}, \Delta, P_{in}, n_0, \overline{M}, \overline{G})$ and a desired nonterminal n_{in} and returns another WRTG, G_{out} that is different from G_{in} in that it has more productions, specifically those beginning with n_{in} that are in $\overline{M}(\overline{G})^{\triangleright}$. Algorithms using stand-ins should call PRODUCE to ensure the stand-in they are using has the desired productions beginning with the specific nonterminal. Note, then, that PRODUCE obtains the effect of forward applica-

Algorithm 4 REPLACE

inputs
  $y \in T_{\Sigma}(X)$
  WRTG $G = (N, \Sigma, P, n_0, \overline{M}, \overline{G})$ in normal form, with no chain productions
  $n \in N$
outputs
  set Π of pairs (θ, w) where θ is a mapping $pos(y) \to N$ and $w \in \mathbb{R}_+^{\infty}$, each pair indicating a successful run on y by productions in G, starting from n, and w is the weight of the run.
complexity
  $O(|P|^{size(y)})$

$\Pi_{last} \leftarrow \{(\{(\varepsilon, n)\}, 1)\}$
for all $v \in pos(y)$ such that $y(v) \notin X$ in prefix order **do**
  $\Pi_v \leftarrow \emptyset$
  for all $(\theta, w) \in \Pi_{last}$ **do**
    if $\overline{M} \neq \emptyset$ **and** $\overline{G} \neq \emptyset$ **then**
      $G \leftarrow$ PRODUCE$(G, \theta(v))$
    for all $(\theta(v) \xrightarrow{w'} y(v)(n_1, \ldots, n_k)) \in P$ **do**
      $\Pi_v \leftarrow \Pi_v \cup \{(\theta \cup \{(vi, n_i), 1 \leq i \leq k\}, w \cdot w')\}$
  $\Pi_{last} \leftarrow \Pi_v$
return Π_{last}

Algorithm 5 MAKE-EXPLICIT

inputs
  WRTG $G = (N, \Sigma, P, n_0, \overline{M}, \overline{G})$ in normal form
outputs
  WRTG $G' = (N', \Sigma, P', n_0, \overline{M}, \overline{G})$, in normal form, such that if $\overline{M} \neq \emptyset$ and $\overline{G} \neq \emptyset$, $L_{G'} = L_{\overline{M}(\overline{G})^{\triangleright}}$, and otherwise $G' = G$.
complexity
  $O(|P'|)$

$G' \leftarrow G$
$\Xi \leftarrow \{n_0\}$ {seen nonterminals}
$\Psi \leftarrow \{n_0\}$ {pending nonterminals}
while $\Psi \neq \emptyset$ **do**
  $n \leftarrow$any element of Ψ
  $\Psi \leftarrow \Psi \setminus \{n\}$
  if $\overline{M} \neq \emptyset$ **and** $\overline{G} \neq \emptyset$ **then**
    $G' \leftarrow$ PRODUCE(G', n)
  for all $(n \xrightarrow{w} \sigma(n_1, \ldots, n_k)) \in P'$ **do**
    for $i = 1$ to k **do**
      if $n_i \notin \Xi$ **then**
        $\Xi \leftarrow \Xi \cup \{n_i\}$
        $\Psi \leftarrow \Psi \cup \{n_i\}$
return G'

g_0
$g_0 \xrightarrow{w_1} \sigma(g_0, g_1)$
$g_0 \xrightarrow{w_2} \alpha \quad g_1 \xrightarrow{w_3} \alpha$

(a) Input WRTG G

a_0
$a_0.\sigma(x_1, x_2) \xrightarrow{w_4} \sigma(a_0.x_1, a_1.x_2)$
$a_0.\sigma(x_1, x_2) \xrightarrow{w_5} \psi(a_2.x_1, a_1.x_2)$
$a_0.\alpha \xrightarrow{w_6} \alpha \quad a_1.\alpha \xrightarrow{w_7} \alpha \quad a_2.\alpha \xrightarrow{w_8} \rho$

(b) First transducer M_A in the cascade

b_0
$b_0.\sigma(x_1, x_2) \xrightarrow{w_9} \sigma(b_0.x_1, b_0.x_2)$
$b_0.\alpha \xrightarrow{w_{10}} \alpha$

(c) Second transducer M_B in the cascade

$g_0a_0 \xrightarrow{w_1 \cdot w_4} \sigma(g_0a_0, g_1a_1)$
$g_0a_0 \xrightarrow{w_1 \cdot w_5} \psi(g_0a_2, g_1a_1)$
$g_0a_0 \xrightarrow{w_2 \cdot w_6} \alpha \quad g_1a_1 \xrightarrow{w_3 \cdot w_7} \alpha$

(d) Productions of $M_A(G)^{\triangleright}$ built as a consequence of building the complete $M_B(M_A(G)^{\triangleright})^{\triangleright}$

$g_0a_0b_0$
$g_0a_0b_0 \xrightarrow{w_1 \cdot w_4 \cdot w_9} \sigma(g_0a_0b_0, g_1a_1b_0)$
$g_0a_0b_0 \xrightarrow{w_2 \cdot w_6 \cdot w_{10}} \alpha \quad g_1a_1b_0 \xrightarrow{w_3 \cdot w_7 \cdot w_{10}} \alpha$

(e) Complete $M_B(M_A(G)^{\triangleright})^{\triangleright}$

Figure 2: Forward application through a cascade of tree transducers using an on-the-fly method.

tion in an on-the-fly manner.[11] It makes calls to REPLACE, which is presented in Algorithm 4, as well as to a NORM algorithm that ensures normal form by replacing a single production not in normal form with several normal-form productions that can be combined together (Alexandrakis and Bozapalidis, 1987) and a CHAIN-REM algorithm that replaces a WRTG containing chain productions with an equivalent WRTG that does not (Mohri, 2009).

As an example of stand-in construction, consider the invocation PRODUCE(G_1, g_0a_0), where G_1 = ($\{g_0a_0\}$, $\{\sigma, \psi, \alpha, \rho\}$, $\emptyset$, g_0a_0, M_A, G), G is in Figure 2a,[12] and M_A is in 2b. The stand-in WRTG that is output contains the first three of the four productions in Figure 2d.

To demonstrate the use of on-the-fly application in a cascade, we next show the effect of PRODUCE when used with the cascade $G \circ M_A \circ M_B$, where M_B is in Figure 2c. Our driving algorithm in this case is Algorithm 5, MAKE-

$r_{\text{JJ}}.\text{JJ}(x_1, x_2, x_3) \rightarrow \text{JJ}(r_{\text{DT}}.x_1, r_{\text{JJ}}.x_2, r_{\text{VB}}.x_3)$
$r_{\text{VB}}.\text{VB}(x_1, x_2, x_3) \rightarrow \text{VB}(r_{\text{NNPS}}.x_1, r_{\text{NN}}.x_3, r_{\text{VB}}.x_2)$
t."gentle" $\rightarrow$ "gentle"

(a) Rotation rules

$i_{\text{VB}}.\text{NN}(x_1, x_2) \rightarrow \text{NN}(\text{INS } i_{\text{NN}}.x_1, i_{\text{NN}}.x_2)$
$i_{\text{VB}}.\text{NN}(x_1, x_2) \rightarrow \text{NN}(i_{\text{NN}}.x_1, i_{\text{NN}}.x_2)$
$i_{\text{VB}}.\text{NN}(x_1, x_2) \rightarrow \text{NN}(i_{\text{NN}}.x_1, i_{\text{NN}}.x_2, \text{INS})$

(b) Insertion rules

$t.\text{VB}(x_1, x_2, x_3) \rightarrow \text{X}(t.x_1, t.x_2, t.x_3)$
t."gentleman" $\rightarrow$ j1
t."gentleman" $\rightarrow$ EPS
t.INS $\rightarrow$ j1
t.INS $\rightarrow$ j2

(c) Translation rules

Figure 3: Example rules from transducers used in decoding experiment. j1 and j2 are Japanese words.

EXPLICIT, which simply generates the full application WRTG using calls to PRODUCE. The input to MAKE-EXPLICIT is G_2 = ($\{g_0a_0b_0\}$, $\{\sigma, \alpha\}$, $\emptyset$, $g_0a_0b_0$, M_B, G_1).[13] MAKE-EXPLICIT calls PRODUCE(G_2, $g_0a_0b_0$). PRODUCE then seeks to cover $b_0.\sigma(x_1, x_2) \xrightarrow{w_9} \sigma(b_0.x_1, b_0.x_2)$ with productions from G_1, which is a stand-in for $M_A(G)^{\triangleright}$. At line 14 of REPLACE, G_1 is improved so that it has the appropriate productions. The productions of $M_A(G)^{\triangleright}$ that must be built to form the complete $M_B(M_A(G)^{\triangleright})^{\triangleright}$ are shown in Figure 2d. The complete $M_B(M_A(G)^{\triangleright})^{\triangleright}$ is shown in Figure 2e. Note that because we used this on-the-fly approach, we were able to avoid building all the productions in $M_A(G)^{\triangleright}$; in particular we did not build $g_0a_2 \xrightarrow{w_2 \cdot w_8} \rho$, while a bucket brigade approach would have built this production. We have also designed an analogous on-the-fly PRODUCE algorithm for backward application on linear WTT.

We have now defined several on-the-fly and bucket brigade algorithms, and also discussed the possibility of embed-compose-project and offline composition strategies to application of cascades of tree transducers. Tables 2a and 2b summarize the available methods of forward and backward application of cascades for recognizability-preserving tree transducer classes.

5 Decoding Experiments

The main purpose of this paper has been to present novel algorithms for performing application. However, it is important to demonstrate these algorithms on real data. We thus demonstrate bucket-brigade and on-the-fly backward application on a typical NLP task cast as a cascade of wLNT. We adapt the Japanese-to-English transla-

[11] Note further that it allows forward application of class wxLNT, something the embed-compose-project approach did not allow.

[12] By convention the initial nonterminal and state are listed first in graphical depictions of WRTGs and WXTTs.

[13] Note that G_2 is the initial stand-in for $M_B(M_A(G)^{\triangleright})^{\triangleright}$, since G_1 is the initial stand-in for $M_A(G)^{\triangleright}$.

method	WST	wxLNT	wLNT
oc	√	×	√
bb	√	×	√
otf	√	√	√

(a) Forward application

method	WST	wxLT	wLT	wxLNT	wLNT
oc	√	×	×	×	√
bb	√	√	√	√	√
otf	√	√	√	√	√

(b) Backward application

Table 2: Transducer types and available methods of forward and backward application of a cascade. oc = offline composition, bb = bucket brigade, otf = on the fly.

tion model of Yamada and Knight (2001) by transforming it from an English-tree-to-Japanese-string model to an English-tree-to-Japanese-*tree* model. The Japanese trees are unlabeled, meaning they have syntactic structure but all nodes are labeled "X". We then cast this modified model as a cascade of LNT tree transducers. Space does not permit a detailed description, but some example rules are in Figure 3. The rotation transducer $\mathcal{R}$, a sample of which is in Figure 3a, has 6,453 rules, the insertion transducer $\mathcal{I}$, Figure 3b, has 8,122 rules, and the translation transducer, $\mathcal{T}$, Figure 3c, has 37,311 rules.

We add an English syntax language model $\mathcal{L}$ to the cascade of transducers just described to better simulate an actual machine translation decoding task. The language model is cast as an identity WTT and thus fits naturally into the experimental framework. In our experiments we try several different language models to demonstrate varying performance of the application algorithms. The most realistic language model is a PCFG. Each rule captures the probability of a particular sequence of child labels given a parent label. This model has 7,765 rules.

To demonstrate more extreme cases of the usefulness of the on-the-fly approach, we build a language model that recognizes exactly the 2,087 trees in the training corpus, each with equal weight. It has 39,455 rules. Finally, to be ultra-specific, we include a form of the "specific" language model just described, but only allow the English counterpart of the particular Japanese sentence being decoded in the language.

The goal in our experiments is to apply a single tree t backward through the cascade $\mathcal{L} \circ \mathcal{R} \circ \mathcal{I} \circ \mathcal{T} \circ t$ and find the 1-best path in the application WRTG. We evaluate the speed of each approach: bucket brigade and on-the-fly. The algorithm we use to obtain the 1-best path is a modification of the k-best algorithm of Pauls and Klein (2009). Our algorithm finds the 1-best path in a WRTG and admits an on-the-fly approach.

The results of the experiments are shown in Table 3. As can be seen, on-the-fly application is generally faster than the bucket brigade, about double the speed per sentence in the traditional

LM type	method	time/sentence
pcfg	bucket	28s
pcfg	otf	17s
exact	bucket	>1m
exact	otf	24s
1-sent	bucket	2.5s
1-sent	otf	.06s

Table 3: Timing results to obtain 1-best from application through a weighted tree transducer cascade, using on-the-fly vs. bucket brigade backward application techniques. pcfg = model recognizes any tree licensed by a pcfg built from observed data, exact = model recognizes each of 2,000+ trees with equal weight, 1-sent = model recognizes exactly one tree.

experiment that uses an English PCFG language model. The results for the other two language models demonstrate more keenly the potential advantage that an on-the-fly approach provides—the simultaneous incorporation of information from all models allows application to be done more effectively than if each information source is considered in sequence. In the "exact" case, where a very large language model that simply recognizes each of the 2,087 trees in the training corpus is used, the final application is so large that it overwhelms the resources of a 4gb MacBook Pro, while the on-the-fly approach does not suffer from this problem. The "1-sent" case is presented to demonstrate the ripple effect caused by using on-the fly. In the other two cases, a very large language model generally overwhelms the timing statistics, regardless of the method being used. But a language model that represents exactly one sentence is very small, and thus the effects of simultaneous inference are readily apparent—the time to retrieve the 1-best sentence is reduced by two orders of magnitude in this experiment.

6 Conclusion

We have presented algorithms for forward and backward application of weighted tree transducer cascades, including on-the-fly variants, and demonstrated the benefit of an on-the-fly approach to application. We note that a more formal approach to application of WTTs is being developed,

independent from these efforts, by Fülöp et al. (2010).

Acknowledgments

We are grateful for extensive discussions with Andreas Maletti. We also appreciate the insights and advice of David Chiang, Steve DeNeefe, and others at ISI in the preparation of this work. Jonathan May and Kevin Knight were supported by NSF grants IIS-0428020 and IIS-0904684. Heiko Vogler was supported by DFG VO 1011/5-1.

References

Athanasios Alexandrakis and Symeon Bozapalidis. 1987. Weighted grammars and Kleene's theorem. *Information Processing Letters*, 24(1):1–4.

Brenda S. Baker. 1979. Composition of top-down and bottom-up tree transductions. *Information and Control*, 41(2):186–213.

Zoltán Ésik and Werner Kuich. 2003. Formal tree series. *Journal of Automata, Languages and Combinatorics*, 8(2):219–285.

Zoltán Fülöp and Heiko Vogler. 2009. Weighted tree automata and tree transducers. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, chapter 9, pages 313–404. Springer-Verlag.

Zoltán Fülöp, Andreas Maletti, and Heiko Vogler. 2010. Backward and forward application of weighted extended tree transducers. Unpublished manuscript.

Ferenc Gécseg and Magnus Steinby. 1984. *Tree Automata*. Akadémiai Kiadó, Budapest.

Liang Huang and David Chiang. 2005. Better k-best parsing. In Harry Bunt, Robert Malouf, and Alon Lavie, editors, *Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT)*, pages 53–64, Vancouver, October. Association for Computational Linguistics.

Werner Kuich. 1998. Formal power series over trees. In Symeon Bozapalidis, editor, *Proceedings of the 3rd International Conference on Developments in Language Theory (DLT)*, pages 61–101, Thessaloniki, Greece. Aristotle University of Thessaloniki.

Werner Kuich. 1999. Tree transducers and formal tree series. *Acta Cybernetica*, 14:135–149.

Andreas Maletti, Jonathan Graehl, Mark Hopkins, and Kevin Knight. 2009. The power of extended top-down tree transducers. *SIAM Journal on Computing*, 39(2):410–430.

Andreas Maletti. 2006. Compositions of tree series transformations. *Theoretical Computer Science*, 366:248–271.

Andreas Maletti. 2008. Compositions of extended top-down tree transducers. *Information and Computation*, 206(9–10):1187–1196.

Andreas Maletti. 2009. Personal Communication.

Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. 2000. The design principles of a weighted finite-state transducer library. *Theoretical Computer Science*, 231:17–32.

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

Mehryar Mohri. 2009. Weighted automata algorithms. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, chapter 6, pages 213–254. Springer-Verlag.

Adam Pauls and Dan Klein. 2009. K-best A* parsing. In Keh-Yih Su, Jian Su, Janyce Wiebe, and Haizhou Li, editors, *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 958–966, Suntec, Singapore, August. Association for Computational Linguistics.

Fernando Pereira and Michael Riley. 1997. Speech recognition by composition of weighted finite automata. In Emmanuel Roche and Yves Schabes, editors, *Finite-State Language Processing*, chapter 15, pages 431–453. MIT Press, Cambridge, MA.

William A. Woods. 1980. Cascaded ATN grammars. *American Journal of Computational Linguistics*, 6(1):1–12.

Kenji Yamada and Kevin Knight. 2001. A syntax-based statistical translation model. In *Proceedings of 39th Annual Meeting of the Association for Computational Linguistics*, pages 523–530, Toulouse, France, July. Association for Computational Linguistics.