Parsing and Translation Algorithms Based on Weighted Extended Tree Transducers

Andreas Maletti*
Departament de Filologies Romàniques
Universitat Rovira i Virgili
Tarragona, Spain

Giorgio Satta
Department of Information Engineering
University of Padua
Padua, Italy

Abstract

This paper proposes a uniform framework for the development of parsing and translation algorithms for weighted extended (top-down) tree transducers and input strings. The asymptotic time complexity of these algorithms can be improved in practice by exploiting an algorithm for rule factorization in the above transducers.

1 Introduction

In the field of statistical machine translation, considerable interest has recently been shown for translation models based on weighted tree transducers. In this paper we consider the so-called weighted extended (top-down) tree transducers (WXTTs for short). WXTTs have been proposed by Graehl and Knight (2004) and Knight (2007) and are rooted in similar devices introduced earlier in the formal language literature (Arnold and Dauchet, 1982).

WXTTs have enough expressivity to represent hierarchical syntactic analyses for natural language sentences and can directly model most of the elementary operations that rule the process of translation between natural languages (Knight, 2007). Furthermore, the use of weights and internal states allows the encoding of statistical parameters that have recently been shown to be extremely useful in discriminating likely translations from less plausible ones.

For an WXTT M, the *parsing* problem is traditionally defined for a pair of trees t and u and requires as output some representation of the set of all computations of M that map t into u. Similarly, the *translation* problem for M is defined for an input tree t and requires as output some representation of the set of all computations of M mapping t into any other tree. When we deal with natural language processing applications, however, parsing and translation are most often represented on the basis of input strings rather than trees. Some tricks are then applied to map the problem back to the case of input trees. As an example in the context of machine translation, let w be some input string to be translated. One can intermediately construct a tree automaton M_w that recognizes the set of all possible trees that have w as yield with internal nodes from the input alphabet of M. This automaton M_w is further transformed into a tree transducer implementing a partial identity translation. This transducer is then composed with M (relational composition) to obtain a transducer that represents all translations of w. This is usually called the 'cascaded' approach.

In contrast with the cascaded approach above, which may be rather inefficient, we investigate a more direct technique for both parsing and translation of strings based on WXTTs. We do this by extending to WXTTs the well-known BAR-HILLEL construction defined for context-free grammars (Bar-Hillel et al., 1964) and for weighted context-free grammars (Nederhof and Satta, 2003). We then derive computational complexity results for parsing and translation of input strings on the basis of WXTTs. Finally, we develop a novel factorization algorithm for WXTTs that, in practical applications, can reduce the asymptotic complexity for such problems.

2 Preliminary definitions

Let $\cdot$ be an associative binary operation on a set S. If S contains an element 1 such that $1 \cdot s = s = s \cdot 1$ for every $s \in S$, then $(S, \cdot, 1)$ is a monoid. Such a monoid $(S, \cdot, 1)$ is commutative if the identity $s_1 \cdot s_2 = s_2 \cdot s_1$ holds for all $s_1, s_2 \in S$. A *commutative semiring* $(S, +, \cdot, 0, 1)$ is an algebraic structure such that:

- $(S, +, 0)$ and $(S, \cdot, 1)$ are commutative

*Financially supported by the *Ministerio de Educación y Ciencia* (MEC) grant JDCI-2007-760.

Proceedings of the 2010 Workshop on Applications of Tree Automata in Natural Language Processing, ACL 2010, pages 19–27,
Uppsala, Sweden, 16 July 2010.

monoids,

- $\cdot$ distributes over $+$ (from both sides), and
- $s \cdot 0 = 0 = 0 \cdot s$ for every $s \in S$.

From now on, let $(S, +, \cdot, 0, 1)$ be a commutative semiring. An *alphabet* is a finite set of symbols. A *weighted string automaton* [WSA] (Schützenberger, 1961; Eilenberg, 1974) is a system $N = (P, \Gamma, J, \nu, F)$ where

- P and Γ are alphabets of states and input symbols, respectively,
- $J, F\colon P \to S$ assign initial and final weights, respectively, and
- $\nu\colon P \times \Gamma \times P \to S$ assigns a weight to each transition.

The transition weight mapping ν can be understood as square matrices $\nu(\cdot, \gamma, \cdot) \in S^{P \times P}$ for every $\gamma \in \Gamma$. The WSA N is *deterministic* if

- $J(p) \neq 0$ for at most one $p \in P$ and
- for every $p \in P$ and $\gamma \in \Gamma$ there exists at most one $p' \in P$ such that $\nu(p, \gamma, p') \neq 0$.

We now proceed with the semantics of N. We will define the *initial algebra semantics* here; alternative, equivalent definitions of the semantics exist (Sakarovitch, 2009). Let $w \in \Gamma^*$ be an input string, $\gamma \in \Gamma$, and $p, p' \in P$ be two states. We extend ν to a mapping $h_\nu\colon P \times \Gamma^* \times P \to S$ recursively as follows:

$$h_\nu(p, \varepsilon, p') = \begin{cases} 1 & \text{if } p = p' \\ 0 & \text{otherwise} \end{cases}$$

$$h_\nu(p, \gamma w, p') = \sum_{p'' \in P} \nu(p, \gamma, p'') \cdot h_\nu(p'', w, p') \ .$$

Consequently,

$$h_\nu(p, uw, p') = \sum_{p'' \in P} h_\nu(p, u, p'') \cdot h_\nu(p'', w, p')$$

for all $p, p' \in P$ and $u, w \in \Gamma^*$. Then the matrix $h_\nu(\cdot, \gamma_1 \cdots \gamma_k, \cdot)$ equals $\nu(\cdot, \gamma_1, \cdot) \cdot \ldots \cdot \nu(\cdot, \gamma_k, \cdot)$. Thus, if the semiring operations can be performed in constant time and access to $\nu(p, \gamma, q)$ is in constant time for every $p, q \in P$, then for every $w \in \Gamma^*$ we can compute the matrix $h_\nu(\cdot, w, \cdot)$ in time $\mathcal{O}(|w| \cdot |P|^3)$ because it can be computed by $|w| - 1$ matrix multiplications.

The WSA N computes the map $N\colon \Gamma^* \to S$, which is defined for every $w \in \Gamma^*$ by[1]

$$N(w) = \sum_{p, p' \in P} J(p) \cdot h_\nu(p, w, p') \cdot F(p') \ .$$

[1]We overload the symbol N to denote both the WSA and its recognized mapping. However, the intended meaning will always be clear from the context.

Since we will also consider individual runs, let us recall the run semantics as well. Let $w = \gamma_1 \cdots \gamma_k \in \Gamma^*$ be an input string of length k. Then any mapping $r\colon [0, k] \to P$ is a *run* of N on w, where $[0, k]$ denotes the set of integers between (inclusive) 0 and k. A run can be understood as a vector of states and thus we sometimes write r_i instead of $r(i)$. The weight of such a run r, denoted by $\mathrm{wt}_N(r)$, is defined by $\mathrm{wt}_N(r) = \prod_{i=1}^{k} \nu(r_{i-1}, \gamma_i, r_i)$. Then

$$h_\nu(p, w, p') = \sum_{\substack{r\colon [0,k] \to P \\ r_0 = p, r_k = p'}} \mathrm{wt}_N(r)$$

for every $p, p' \in P$ and $w \in \Gamma^*$.

3 Weighted extended tree transducers

Next, we move to tree languages, for which we need to introduce some additional notation. Let Σ be a *ranked alphabet*, that is, an alphabet whose symbols have a unique associated arity. We write Σ_k to denote the set of all k-ary symbols in Σ. We use the special nullary symbol $e \in \Sigma_0$ to syntactically represent the empty string ε. The set of Σ-*trees* indexed by a set V, denoted by $T_\Sigma(V)$, is the smallest set satisfying both of the following conditions:

- for every $v \in V$, the single node labeled v, written v, is a tree of $T_\Sigma(V)$,
- for every $\sigma \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma(V)$, the tree with a root node labeled σ and trees $t_1, \ldots, t_k$ as its k children, written $\sigma(t_1, \ldots, t_k)$, belongs to $T_\Sigma(V)$.

Throughout this paper we sometimes write $\sigma()$ as just σ. In the following, let $t \in T_\Sigma(V)$. The set of *positions* $\mathrm{Pos}(t) \subseteq \mathbb{N}^*$ of a tree $t \in T_\Sigma(V)$ is recursively defined as follows:

$$\mathrm{Pos}(v) = \{\varepsilon\}$$
$$\mathrm{Pos}(t) = \{\varepsilon\} \cup \{iw \mid 1 \leq i \leq k, w \in \mathrm{Pos}(t_i)\}$$

for every $v \in V$, $\sigma \in \Sigma_k$, and $t_1, \ldots, t_k \in T_\Sigma(V)$ where $t = \sigma(t_1, \ldots, t_k)$. The label of t at position $w \in \mathrm{Pos}(t)$ is denoted by $t(w)$. The size of the tree $t \in T_\Sigma$ is defined as $|t| = |\mathrm{Pos}(t)|$. For every $w \in \mathrm{Pos}(t)$ the subtree of t that is rooted at w is denoted by $\mathrm{sub}_t(w)$; i.e.,

$$\mathrm{sub}_t(\varepsilon) = t$$
$$\mathrm{sub}_{\sigma(t_1, \ldots, t_k)}(iw) = \mathrm{sub}_{t_i}(w)$$

for every $\sigma \in \Sigma_k$, $t_1, \ldots, t_k \in T_\Sigma(V)$, $1 \leq i \leq k$, and $w \in \mathrm{Pos}(t_i)$. Finally, the set of variables $\mathrm{var}(t)$ is given by

$$\mathrm{var}(t) = \{v \in V \mid \exists w \in \mathrm{Pos}(t) \colon t(w) = v\} \ .$$

If for every $v \in \mathrm{var}(t)$ there exists exactly one $w \in \mathrm{Pos}(t)$ such that $t(w) = v$, then t is *linear*.

We use the fixed sets $X = \{x_i \mid i \geq 1\}$ and $Y = \{y_{i,j} \mid 1 \leq i < j\}$ of formal variables and the subsets $X_k = \{x_i \mid 1 \leq i \leq k\}$ and $Y_k = \{y_{i,j} \mid 1 \leq i < j \leq k\}$ for every $k \geq 0$. Note that $X_0 = \emptyset$. For every $H \subseteq \Sigma_0 \cup X \cup Y$, the H-yield of t is recursively defined by $\mathrm{yd}_H(t) = t$ if $t \in H \setminus \{e\}$, $\mathrm{yd}_H(t) = \mathrm{yd}_H(t_1) \cdots \mathrm{yd}_H(t_k)$ if $t = \sigma(t_1, \ldots, t_k)$ with $\sigma \in \Sigma_k$ and $k \geq 1$, and $\mathrm{yd}_H(t) = \varepsilon$ otherwise. If $H = \Sigma_0 \cup X \cup Y$, then we also omit the index and just write $\mathrm{yd}(t)$.

Let $l \in T_\Sigma(V)$ and $\theta \colon V \to T_\Sigma(V)$. Then $l\theta$ denotes the result obtained from l by replacing every occurrence of $v \in V$ by $\theta(v)$. The k-fold application is denoted by $l\theta^k$. If $l\theta^k = l\theta^{k+1}$ for some $k \geq 0$, then we denote $l\theta^k$ by $l\theta^*$. In addition, if $V = X_k$, then we write $l[\theta(x_1), \ldots, \theta(x_k)]$ instead of $l\theta$. We write $C_\Sigma(X_k)$ for the subset of those trees of $T_\Sigma(X_k)$ such that every variable of $x \in X_k$ occurs exactly once in it. Given $t \in T_\Sigma(X)$, we write $\mathrm{dec}(t)$ for the set

$$\left\{(l, t_1, \ldots, t_k) \middle| \begin{array}{l} l \in C_\Sigma(X_k), l[t_1, \ldots, t_k] = t, \\ t_1, \ldots, t_k \in T_\Sigma(X) \end{array} \right\}$$

A *(linear and nondeleting) weighted extended (top-down) tree transducer* [WXTT] (Arnold and Dauchet, 1975; Arnold and Dauchet, 1976; Lilin, 1981; Arnold and Dauchet, 1982; Maletti et al., 2009) is a system $M = (Q, \Sigma, \Delta, I, R)$ where

- Q is an alphabet of states,
- Σ and Δ are ranked alphabets of input and output symbols, respectively,
- $I \colon Q \to S$ assigns initial weights, and
- R is a finite set of rules of the form $(q, l) \xrightarrow{s} (q_1 \cdots q_k, r)$ with $q, q_1, \ldots, q_k \in Q$, $l \in C_\Sigma(X_k)$ and $r \in C_\Delta(X_k)$, and $s \in S$ such that $\{l, r\} \not\subseteq X$.

Let us discuss the final restriction imposed on the rules of a WXTT. Essentially, it disallows rules of the form $(q, x_1) \xrightarrow{s} (q', x_1)$ with $q, q' \in Q$ and $s \in S$. Such *pure epsilon rules* only change the state and charge a cost. However, they can yield infinite derivations (and with it infinite products and sums) and are not needed in our applications. The WXTT M is *standard* if $\mathrm{yd}_X(r) = x_1 \cdots x_k$ for every $(q, l) \xrightarrow{s} (q_1 \cdots q_k, r) \in R$. This restriction enforces that the order of the variables is fixed on the right-hand side r, but since the order is arbitrary in the left-hand side l (and the names of the variables are inconsequential), it can be achieved easily without loss of generality. If there are several rules that differ only in the naming of the variables, then their weights should be added to obtain a single standard rule. To keep the presentation simple, we also construct nonstandard WXTTs in the sequel. However, we implicitly assume that those are converted into standard WXTTs.

The semantics of a standard WXTT is inspired by the initial-algebra semantics for classical weighted top-down and bottom-up tree transducers (Fülöp and Vogler, 2009) [also called top-down and bottom-up tree series transducers by Engelfriet et al. (2002)]. Note that our semantics is equivalent to the classical term rewriting semantics, which is presented by Graehl and Knight (2004) and Graehl et al. (2008), for example. In fact, we will present an equivalent semantics based on runs later. Let $M = (Q, \Sigma, \Delta, I, R)$ be a WXTT. We present a definition that is more general than immediately necessary, but the generalization will be useful later on. For every $n \in \mathbb{N}$, $p_1, \ldots, p_n \in Q$, and $L \subseteq R$, we define the mapping $h_L^{p_1 \cdots p_n} \colon T_\Sigma(X_n) \times T_\Delta(X_n) \to S^Q$ by $h_L^{p_1 \cdots p_n}(x_i, x_i)_{p_i} = 1$ for every $1 \leq i \leq n$ and

$$h_L^{p_1 \cdots p_n}(t, u)_q = \sum_{\substack{(l, t_1, \ldots, t_k) \in \mathrm{dec}(t) \\ (r, u_1, \ldots, u_k) \in \mathrm{dec}(u) \\ (q, l) \xrightarrow{s} (q_1 \cdots q_k, r) \in L}} s \cdot \prod_{i=1}^{k} h_L^{p_1 \cdots p_n}(t_i, u_i)_{q_i} \quad (1)$$

for all remaining $t \in T_\Sigma(X_n)$, $u \in T_\Delta(X_n)$, and $q \in Q$. Note that for each nonzero summand in (1) one of the decompositions $\mathrm{dec}(t)$ and $\mathrm{dec}(u)$ must be proper (i.e., either $l \notin X$ or $r \notin X$). This immediately yields that the sum is finite and the recursion well-defined. The transformation computed by M, also denoted by M, is the mapping $M \colon T_\Sigma \times T_\Delta \to S$, which is defined by $M(t, u) = \sum_{q \in Q} I(q) \cdot h_R(t, u)_q$ for every $t \in T_\Sigma$ and $u \in T_\Delta$.

Let us also introduce a run semantics for the WXTT $(Q, \Sigma, \Delta, I, R)$. The rank of a rule $\rho = (q, l) \xrightarrow{s} (q_1 \cdots q_k, r) \in R$, denoted by $\mathrm{rk}(\rho)$, is $\mathrm{rk}(\rho) = k$. This turns R into a ranked alphabet. The input state of ρ is $\mathrm{in}(\rho) = q$, the ith output state is $\mathrm{out}_i(\rho) = q_i$ for every $1 \leq i \leq k$, and

the weight of ρ is $\mathrm{wt}(\rho) = s$. A tree $r \in T_R(X)$ is called *run* if $\mathrm{in}(r(wi)) = \mathrm{out}_i(r(w))$ for every $wi \in \mathrm{Pos}(r)$ and $1 \leq i \leq \mathrm{rk}(r(w))$ such that $r(wi) \in R$. The weight of a run $r \in T_R(X)$ is

$$\mathrm{wt}(r) = \prod_{w \in \mathrm{Pos}(r), r(w) \in R} \mathrm{wt}(r(w)) \ .$$

The evaluation mappings $\pi_1 \colon T_R(X) \to T_\Sigma(X)$ and $\pi_2 \colon T_R(X) \to T_\Delta(X)$ are defined for every $x \in X$, $\rho = (q, l) \xrightarrow{s} (q_1 \cdots q_k, r) \in R$, and $r_1, \ldots, r_k \in T_R(X)$ by $\pi_1(x) = x$, $\pi_2(x) = x$, and

$$\pi_1(\rho(r_1, \ldots, r_k)) = l[\pi_1(r_1), \ldots, \pi_1(r_k)]$$
$$\pi_2(\rho(r_1, \ldots, r_k)) = r[\pi_2(r_1), \ldots, \pi_2(r_k)] \ .$$

We obtain the weighted tree transformation for every $t \in T_\Sigma$ and $u \in T_\Delta$ as follows[2]

$$M(t, u) = \sum_{\substack{\text{run } r \in T_R \\ t = \pi_1(r), u = \pi_2(r)}} I(\mathrm{in}(r(\varepsilon))) \cdot \mathrm{wt}(r) \ .$$

This approach is also called the *bimorphism approach* (Arnold and Dauchet, 1982) to tree transformations.

4 Input and output restrictions of WXTT

In this section we will discuss the BAR-HILLEL construction for the input and the output part of a WXTT M. This construction essentially restricts the input or output of the WXTT M to the string language recognized by a WSA N. Contrary to (direct or inverse) application, this construction is supposed to yield another WXTT. More precisely, the constructed WXTT should assign to each translation (t, u) the weight assigned to it by M multiplied by the weight assigned by N to the yield of t (or u if the output is restricted). Since our WXTTs are symmetric, we will actually only need one construction. Let us quickly establish the mentioned symmetry statement. Essentially we just have to exchange left- and right-hand sides and redistribute the states in those left- and right-hand sides accordingly.

From now on, let $M = (Q, \Sigma, \Delta, I, R)$ be a WXTT.

Theorem 1. *There exists a WXTT M' such that $M'(u, t) = M(t, u)$ for every $t \in T_\Sigma$ and $u \in T_\Delta$.*

[2]We immediately also use M for the run semantics because the two semantics trivially coincide.

Proof. Let $M' = (Q, \Delta, \Sigma, I, R')$ be the WXTT such that

$$R' = \{(q, r) \xrightarrow{s} (w, l) \mid (q, l) \xrightarrow{s} (w, r) \in R\} \ .$$

It should be clear that $M'(u, t) = M(t, u)$ for every $t \in T_\Sigma$ and $u \in T_\Delta$. □

With the symmetry established, we now only need to present the BAR-HILLEL construction for either the input or output side. Without loss of generality, let us assume that M is standard. We then choose the output side here because the order of variables is fixed in it. Note that we sometimes use the angled parentheses '$\langle$' and '$\rangle$' instead of parentheses for clarity.

Definition 2. *Let $N = (P, \Gamma, J, \nu, F)$ be a WSA with $\Gamma = \Delta_0 \setminus \{e\}$. We construct the output product $\mathrm{Prod}(M, N) = (P \times Q \times P, \Sigma, \Delta, I', R')$ such that*

- *$I'(\langle p, q, p' \rangle) = J(p) \cdot I(q) \cdot F(p')$ for every $p, p' \in P$ and $q \in Q$,*
- *for every rule $(q, l) \xrightarrow{s} (q_1 \cdots q_k, r) \in R$ and every $p_0, \ldots, p_k, p'_0, \ldots, p'_k \in P$, let*

$$(q', l) \xrightarrow{s \cdot s_0 \cdot \ldots \cdot s_k} (q'_1 \cdots q'_k, r) \in R'$$

where
  - *$q' = \langle p_0, q, p'_k \rangle$,*
  - *$q'_i = \langle p'_{i-1}, q_i, p_i \rangle$ for every $1 \leq i \leq k$,*
  - *$\mathrm{yd}(r) = w_0 x_1 w_1 \cdots w_{k-1} x_k w_k$ with $w_0, \ldots, w_k \in \Gamma^*$, and*
  - *$s_i = h_\nu(p_i, w_i, p'_i)$ for every $0 \leq i \leq k$.*

Let $\rho = (q, l) \xrightarrow{s} (q_1 \cdots q_k, r) \in R$. The *size* of ρ is $|\rho| = |l| + |r|$. The size and rank of the WXTT M are $|M| = \sum_{\rho \in R} |\rho|$ and $\mathrm{rk}(M) = \max_{\rho \in R} \mathrm{rk}(\rho)$, respectively. Finally, the maximal output yield length of M, denoted by $\mathrm{len}(M)$, is the maximal length of $\mathrm{yd}(r)$ for all rules $(q, l) \xrightarrow{s} (q_1 \cdots q_k, r) \in R$. The size and rank of $\mathrm{Prod}(M, N)$ are in $\mathcal{O}(|M| \cdot |P|^{2\,\mathrm{rk}(M)+2})$ and $\mathrm{rk}(M)$, respectively. We can compute $\mathrm{Prod}(M, N)$ in time $\mathcal{O}(|R| \cdot \mathrm{len}(M) \cdot |P|^{2\,\mathrm{rk}(M)+5})$. If N is deterministic, then the size of $\mathrm{Prod}(M, N)$ is in $\mathcal{O}(|M| \cdot |P|^{\mathrm{rk}(M)+1})$ and the required time is in $\mathcal{O}(|R| \cdot \mathrm{len}(M) \cdot |P|^{\mathrm{rk}(M)+1})$. Next, let us prove that our BAR-HILLEL construction is actually correct.

Theorem 3. *Let M and N be as in Definition 2, and let $M' = \mathrm{Prod}(M, N)$. Then $M'(t, u) = M(t, u) \cdot N(\mathrm{yd}(u))$ for every $t \in T_\Sigma$ and $u \in T_\Delta$.*

Proof. Let $M' = (Q', \Sigma, \Delta, I', R')$. First, a simple proof shows that

$$h_{R'}(t,u)_{\langle p,q,p'\rangle} = h_R(t,u)_q \cdot h_\nu(p, \mathrm{yd}(u), p')$$

for every $t \in T_\Sigma$, $u \in T_\Delta$, $q \in Q$, and $p, p' \in P$. Now we can prove the main statement as follows:

$$\begin{aligned} & M'(t,u) \\ &= \sum_{q' \in Q'} I'(q') \cdot h_{R'}(t,u)_{q'} \\ &= \sum_{\substack{p,p' \in P \\ q \in Q}} I'(\langle p,q,p'\rangle) \cdot h_R(t,u)_q \cdot h_\nu(p, \mathrm{yd}(u), p') \\ &= M(t,u) \cdot N(\mathrm{yd}(u)) \end{aligned}$$

for every $t \in T_\Sigma$ and $u \in T_\Delta$. □

Note that the typical property of many BAR-HILLEL constructions, namely that a run of M and a run of N uniquely determine a run of $\mathrm{Prod}(M,N)$ and vice versa, does not hold for our construction. In fact, a run of M and a run of N uniquely determine a run of $\mathrm{Prod}(M,N)$, but the converse does not hold. We could modify the construction to enable this property at the expense of an exponential increase in the number of states of $\mathrm{Prod}(M,N)$. However, since those relations are important for our applications, we explore the relation between runs in some detail here.

To simplify the discussion, we assume, without loss of generality, that M is standard and $s = s'$ for every two rules $(q,l) \xrightarrow{s} (w,r) \in R$ and $(q,l) \xrightarrow{s'} (w,r) \in R$. Moreover, we assume the symbols of Definition 2. For every $r' \in T_{R'}(X)$, we let $\mathrm{base}(r')$ denote the run obtained from r' by replacing each symbol

$$(q',l) \xrightarrow{s \cdot s_0 \cdot \ldots \cdot s_k} (q'_1 \cdots q'_k, r)$$

by just $(q,l) \xrightarrow{s} (q_1 \cdots q_k, r) \in R$. Thus, we replace a rule (which is a symbol) of R' by the underlying rule of R. We start with a general lemma, which we believe to be self-evident.

Lemma 4. *Let $r' \in T_{R'}$ and $n = |\mathrm{yd}(\pi_2(r'))|$. Then $\mathrm{wt}_{M'}(r') = \mathrm{wt}_M(\mathrm{base}(r')) \cdot \sum_{r \in R''} \mathrm{wt}_N(r)$ where R'' is a nonempty subset of $\{r\colon [0,n] \to P \mid \mathrm{in}(r'(\varepsilon)) = \langle r_0, q, r_n\rangle\}$.*

Let us assume that N is trim (i.e., all states are reachable and co-reachable) and unambiguous. In this case, for every $\gamma_1 \cdots \gamma_k \in \Gamma^*$ and $p, p' \in P$ there is at most one successful run $r\colon [0,k] \to P$ such that

- $\nu(r_{i-1}, \gamma_i, r_i) \neq 0$ for every $1 \leq i \leq k$, and
- $r_0 = p$ and $r_k = p'$.

This immediately yields the following corollary.

Corollary 5 (of Lemma 4). *Let N be trim and unambiguous. For every $r' \in T_{R'}$ we have*

$$\mathrm{wt}_{M'}(r') = \mathrm{wt}_M(\mathrm{base}(r')) \cdot \mathrm{wt}_N(r)$$

for some $r\colon [0,n] \to P$ with $n = |\mathrm{yd}(\pi_2(r'))|$.

We now turn to applications of the product construction. We first consider the translation problem for an input string w and a WXTT M. We can represent w as a trim and unambiguous WSA N_w that recognizes the language $\{w\}$ with weight of 1 on each transition (which amounts to ignoring the weight contribution of N_w). Then the input product transducer $M_w = \mathrm{Prod}(N_w, M)$ provides a compact representation of the set of all computations of M that translate the string w. From Corollary 5 we have that the weights of these computations are also preserved. Thus, $M_w(T_\Sigma \times T_\Delta) = \sum_{(t,u) \in T_\Sigma \times T_\Delta} M_w(t,u)$ is the weight of the set of string translations of w.

As usual in natural language processing applications, we can exploit appropriate semirings and compute several useful statistical parameters through $M_w(T_\Sigma \times T_\Delta)$, as for instance the highest weight of a computation, the inside probability and the rule expectations; see (Li and Eisner, 2009) for further discussion.

One could also construct in linear time the range tree automaton for M_w, which can be interpreted as a parsing forest with all the weighted trees assigned to translations of w under M. If we further assume that M is unambiguous, then M_w will also have this property, and we can apply standard techniques to extract from M_w the highest score computation. In machine translation applications, the unambiguity assumption is usually met, and avoids the so-called 'spurious' ambiguity, that is, having several computations for an individual pair of trees.

The parsing problem for input strings w and u can be treated in a similar way, by restricting M both to the left and to the right.

5 Rule factorization

As already discussed, the time complexity of the product construction is an exponential function of the rank of the transducer. Unfortunately, it is not possible in the general case to cast a

WXTT into a normal form such that the rank is bounded by some constant. This is also expected from the fact that the translation problem for subclasses of WXTTs such as synchronous context-free grammars is NP-hard (Satta and Peserico, 2005). Nonetheless, there are cases in which a rank reduction is possible, which might result in an improvement of the asymptotical run-time of our construction.

Following the above line, we present here a linear time algorithm for reducing the rank of a WXTT under certain conditions. Similar algorithms for tree-based transformation devices have been discussed in the literature. Nesson et al. (2008) consider synchronous tree adjoining grammars; their algorithm is conceptually very similar to ours, but computationally more demanding due to the treatment of adjunction. Following that work, we also demand here that the new WXTT 'preserves' the recursive structure of the input WXTT, as formalized below. Galley et al. (2004) algorithm also behaves in linear time, but deals with the different problem of tree to string translation. Rank reduction algorithms for string-based translation devices have also been discussed by Zhang et al. (2006) and Gildea et al. (2006).

Recall that $M = (Q, \Sigma, \Delta, I, R)$ is a standard WXTT. Let $M' = (Q', \Sigma, \Delta, I', R')$ be a WXTT with $Q \subseteq Q'$.[3] Then M' is a *structure-preserving factorization* of M if

- $I'(q) = I(q)$ for every $q \in Q$ and $I'(q) = 0$ otherwise, and
- $h_{R'}^{p_1 \cdots p_n}(t,u)_q = h_R^{p_1 \cdots p_n}(t,u)_q$ for every $q, p_1, \ldots, p_n \in Q$, $t \in T_\Sigma(X_n)$, and $u \in T_\Delta(X_n)$.

In particular, we have $h_{R'}(t,u)_q = h_R(t,u)_q$ for $n = 0$. Consequently, M' and M are equivalent because

$$\begin{aligned} M'(t,u) &= \sum_{q \in Q'} I'(q) \cdot h_{R'}(t,u)_q \\ &= \sum_{q \in Q} I(q) \cdot h_R(t,u)_q = M(t,u) \ . \end{aligned}$$

Note that the relation 'is structure-preserving factorization of' is reflexive and transitive, and thus, a pre-order. Moreover, in a ring (actually, additively cancellative semirings are sufficient) it is also anti-symmetric, and consequently, a partial order.

[3] Actually, an injective mapping $Q \to Q'$ would be sufficient, but since the naming of the states is arbitrary, we immediately identify according to the injective mapping.

Informally, a structure-preserving factorization of M consists in a set of new rules that can be composed to provide the original rules and preserve their weights. We develop an algorithm for finding a structure-preserving factorization by decomposing each rule as much as possible. The algorithm can then be iterated for all the rules in the WXTT. The idea underlying our algorithm is very simple. Let $\rho = (q, l) \xrightarrow{s} (q_1 \cdots q_k, r) \in R$ be an original rule. We look for subtrees l' and r' of l and r, respectively, such that $\mathrm{var}(l') = \mathrm{var}(r')$. The condition that $\mathrm{var}(l') = \mathrm{var}(r')$ is derived from the fact that $h_R^{q_1 \cdots q_k}(l', r')_q = 0$ if $\mathrm{var}(l') \neq \mathrm{var}(r')$. We then split ρ into two new rules by 'excising' subtrees l' and r' from l and r, respectively. In the remaining trees the 'excised' trees are replaced with some fresh variable. The tricky part is the efficient computation of the pairs (w_l, w_r), since in the worst case the number of such pairs is in $\Theta(|l| \cdot |r|)$, and naïve testing of the condition $\mathrm{var}(l') = \mathrm{var}(r')$ takes time $\mathcal{O}(\mathrm{rk}(\rho))$.

Let us start with the formal development. Recall the doubly-indexed set $Y = \{y_{i,j} \mid 1 \leq i < j\}$. Intuitively speaking, the variable $y_{i,j}$ will represent the set $\{x_i, \ldots, x_j\}$. With this intuition in mind, we define the mapping $\mathrm{vars}\colon T_\Sigma(X \cup Y) \to \mathbb{N}_\infty^3$ as follows:

$$\begin{aligned} \mathrm{vars}(x_i) &= (i, i, 1) \\ \mathrm{vars}(y_{i,j}) &= (i, j, j - i + 1) \end{aligned}$$

and $\mathrm{vars}(\sigma(t_1, \ldots, t_k))$ is

$$(\min_{\ell=1}^{k} \mathrm{vars}(t_\ell)_1, \max_{\ell=1}^{k} \mathrm{vars}(t_\ell)_2, \sum_{\ell=1}^{k} \mathrm{vars}(t_\ell)_3)$$

for every $i, j \in \mathbb{N}$ with $i < j$, $\sigma \in \Sigma_k$, and $t_1, \ldots, t_k \in T_\Sigma(X \cup Y)$. Clearly, $\mathrm{vars}(t)$ can be computed in time $\mathcal{O}(|t|)$, which also includes the computation of $\mathrm{vars}(u)$ for every subtree u of t. In addition, $\mathrm{vars}(t)_3 = |\mathrm{var}(t)|$ for all linear $t \in T_\Sigma(X)$. Finally, if $t \in T_\Sigma(X)$, then $\mathrm{vars}(t)_1$ and $\mathrm{vars}(t)_2$ are the minimal and maximal index $i \in \mathbb{N}$ such that $x_i \in \mathrm{var}(t)$, respectively (they are ∞ and 0, respectively, if $\mathrm{var}(t) = \emptyset$). For better readability, we use $\mathrm{minvar}(t)$ and $\mathrm{maxvar}(t)$ for $\mathrm{vars}(t)_1$ and $\mathrm{vars}(t)_2$, respectively.

Let $\rho = (q, l) \xrightarrow{s} (q_1 \cdots q_k, r) \in R$ be an original rule. In the following, we will use $\mathrm{minvar}(t)$, $\mathrm{maxvar}(t)$, and $|\mathrm{var}(t)|$ freely for all subtrees t of l and r and assume that they are precomputed,

which can be done in time $\mathcal{O}(|\rho|)$. Moreover, we will freely use the test '$\mathrm{var}(t) = \mathrm{var}(u)$' for subtrees t and u of l and r, respectively. This test can be performed in constant time [disregarding the time needed to precompute $\mathrm{vars}(t)$ and $\mathrm{vars}(u)$] by the equivalent test

- $\mathrm{minvar}(t) = \mathrm{minvar}(u)$,
- $\mathrm{maxvar}(t) = \mathrm{maxvar}(u)$,
- $|\mathrm{var}(t)| = \mathrm{maxvar}(t) - \mathrm{minvar}(t) + 1$, and
- $|\mathrm{var}(u)| = \mathrm{maxvar}(u) - \mathrm{minvar}(u) + 1$.

Our factorization algorithm is presented in Algorithm 1. Its first two parameters hold the left- and right-hand side (l, r), which are to be decomposed. The third and fourth parameter should initially be x_1. To simplify the algorithm, we assume that it is only called with left- and right-hand sides that (i) contain the same variables and (ii) contain at least two variables. These conditions are ensured by the algorithm for the recursive calls. The algorithm returns a decomposition of (l, r) in the form of a set $\mathcal{D} \subseteq T_\Sigma(X \cup Y) \times T_\Delta(X \cup Y)$ such that $\mathrm{var}(l') = \mathrm{var}(r')$ for every $(l', r') \in \mathcal{D}$. Moreover, all such l' and r' are linear. Finally, the pairs in $\mathcal{D}$ can be composed (by means of pointwise substitution at the variables of Y) to form the original pair (l, r).

Before we move on to formal properties of Algorithm 1, let us illustrate its execution on an example.

Example 6. *We work with the left-hand side $l = \sigma(x_1, \sigma(x_3, x_2))$ and the right-hand side $r = \gamma(\sigma(x_1, \gamma(\sigma(x_2, x_3))))$. Then $|\mathrm{var}(l)| \geq 2$ and $\mathrm{var}(l) = \mathrm{var}(r)$. Let us trace the call $\textsc{Decompose}(l, r, x_1, x_1)$. The condition in line 1 is clearly false, so we proceed with line 3. The condition is true for $i = 1$, so we continue with $\textsc{Decompose}(l, \sigma(x_1, \gamma(\sigma(x_2, x_3))), x_1, \gamma(x_1))$.*

This time neither the condition in line 1 nor the condition in line 3 are true. In line 6, j is set to 1 and we initialize $r'_1 = x_1$ and $r'_2 = \gamma(\sigma(x_2, x_3))$. Moreover, the array h is initialized to $h(1) = 1$, $h(2) = 2$, and $h(3) = 2$. Now let us discuss the main loop starting in line 12 in more detail. First, we consider $i = 1$. Since $l_1 = x_1$, the condition in line 13 is fulfilled and we set $l'_1 = x_1$ and proceed with the next iteration ($i = 2$). This time the condition of line 13 is false because $l_2 = \sigma(x_3, x_2)$ and $\mathrm{var}(l_2) = \mathrm{var}(r_{h(2)}) = \mathrm{var}(r_2) = \{x_2, x_3\}$. Consequently, j is set to 2 and $l'_2 = r'_2 = y_{2,3}$. Next, $\textsc{Decompose}(\sigma(x_3, x_2), \gamma(\sigma(x_2, x_3)), x_1, x_1)$ is processed. Let us suppose that it generates the set $\mathcal{D}$. Then we return

$$\mathcal{D} \cup \{(\sigma(x_1, y_{2,3}), \gamma(\sigma(x_1, y_{2,3})))\} \ .$$

Finally, let us quickly discuss how the set $\mathcal{D}$ is obtained. Since the condition in line 3 is true, we have to evaluate the recursive call $\textsc{Decompose}(\sigma(x_3, x_2), \sigma(x_2, x_3), x_1, \gamma(x_1))$.

Now, $j = 2$, $h(2) = 1$, and $h(3) = 2$. Moreover, $r'_1 = x_2$ and $r'_2 = x_3$. In the main loop starting in line 12, the condition of line 13 is always fulfilled, which yields that $l'_1 = x_3$ and $l'_2 = x_2$. Thus, we return $\{(\sigma(x_3, x_2), \gamma(\sigma(x_2, x_3)))\}$, which is exactly the input because decomposition completely failed. Thus, the overall decomposition of l and r is

$$\{(\sigma(x_1, y_{2,3}), \gamma(\sigma(x_1, y_{2,3}))), \\ (\sigma(x_3, x_2), \gamma(\sigma(x_2, x_3)))\} \ ,$$

which, when the second pair is substituted (pointwise) for $y_{2,3}$ in the first pair, yields exactly (l, r).

Informally, the rules are obtained as follows from $\mathcal{D}$. If all variables occur in a pair $(l', r') \in \mathcal{D}$, then the left-hand side is assigned to the original input state. Furthermore, for every variable $y_{i,j}$ we introduce a new fresh state $q_{i,j}$ whereas the variable x_i is associated to q_i. In this way, we determine the states in the right-hand side.

Formally, let $\rho = (q, l) \xrightarrow{s} (q_1 \cdots q_k, r)$ be the original rule and $\mathcal{D}$ be the result of $\textsc{Decompose}(l, r, x_1, x_1)$ of Algorithm 1. In addition, for every $1 \leq i < j \leq k$, let $q_{\rho,i,j}$ be a new state such that $q_{\rho,1,k} = q$. Let

$$Q'_\rho = \{q, q_1, \ldots, q_k\} \cup \{q_{\rho,i,j} \mid 1 \leq i < j \leq k\} \ .$$

Then for every $(l', r') \in \mathcal{D}$ we obtain the rule

$$(q_{\rho,\mathrm{minvar}(r'),\mathrm{maxvar}(r')}, l') \xrightarrow{s'} (p_1 \cdots p_n, r')$$

where $\mathrm{yd}_{X \cup Y}(r') = z_1 \cdots z_n$,

$$s' = \begin{cases} s & \text{if } \mathrm{vars}(r')_3 = k \\ 1 & \text{otherwise} \end{cases}$$

$$q'_\ell = \begin{cases} q_j & \text{if } z_\ell = x_j \\ q_{\rho,i,j} & \text{if } z_\ell = y_{i,j} \end{cases}$$

for every $1 \leq \ell \leq n$. The rules obtained in this fashion are collected in R'_ρ.[4] The WXTT $\mathrm{dec}(M)$ is $\mathrm{dec}(M) = (Q', \Sigma, \Delta, I', R')$ where

[4] Those rules need to be normalized to obtain a standard WXTT.

Algorithm 1 DECOMPOSE(l, r, l', r') computing the decomposition of linear $l \in T_\Sigma(X_k)$ and $r \in T_\Delta(X_k)$ with $\mathrm{var}(l) = \mathrm{var}(r)$ and $|\mathrm{var}(l)| \geq 2$.

```
    if l = σ(l_1, ..., l_m) and there exists i ∈ ℕ is such that var(l_i) = var(l) then
 2:     return DECOMPOSE(l_i, r, l'[σ(l_1, ..., l_{i-1}, x_1, l_{i+1}, ..., l_m)], r'[x_1])
    if r = δ(r_1, ..., r_n) and there exists i ∈ ℕ is such that var(r_i) = var(r) then
 4:     return DECOMPOSE(l, r_i, l'[x_1], r'[δ(r_1, ..., r_{i-1}, x_1, r_{i+1}, ..., r_n)])

    let l = σ(l_1, ..., l_m) and r = δ(r_1, ..., r_n)
 6: j = minvar(r)
    for all 1 ≤ i ≤ n do
 8:     r'_i = r_i
        while j ≤ maxvar(r_i) do
10:         h(j) = i; j = j + 1
    D = ∅
12: for all 1 ≤ i ≤ m do
        if |var(l_i)| ≤ 1 or var(l_i) ≠ var(r_{h(minvar(l_i))}) then
14:         l'_i = l_i
        else
16:         j = h(minvar(l_i))
            l'_i = r'_j = y_{minvar(l_i),maxvar(l_i)}
18:         D = D ∪ DECOMPOSE(l_i, r_j, x_1, x_1)
    return D ∪ {(l'[σ(l'_1, ..., l'_m)], r'[δ(r'_1, ..., r'_n)])}
```

- $Q' = Q \cup \bigcup_{\rho \in R, \mathrm{rk}(\rho) \geq 2} Q'_\rho$,
- $I'(q) = I(q)$ for every $q \in Q$ and $I'(q) = 0$ otherwise, and
- R' is

$$\{\rho \in R \mid \mathrm{rk}(\rho) < 2\} \cup \bigcup_{\rho \in R, \mathrm{rk}(\rho) \geq 2} R'_\rho \ .$$

To measure the success of the factorization, we introduce the following notion. The degree of M, denoted by $\deg(M)$, is the minimal rank of all structure-preserving factorizations M' of M; i.e.,

$$\deg(M) = \min_{\substack{M' \text{ a structure-preserving} \\ \text{factorization of } M}} \mathrm{rk}(M') \ .$$

Then the goal of this section is the efficient computation of a structure-preserving factorization M' of M such that $\mathrm{rk}(M') = \deg(M)$.

Theorem 7. *The WXTT* $\mathrm{dec}(M)$ *is a structure-preserving factorization of* M *such that* $\mathrm{rk}(\mathrm{dec}(M)) = \deg(M)$. *Moreover,* $\mathrm{dec}(M)$ *can be computed in time* $\mathcal{O}(|M|)$.

Proof. Let us only discuss the run-time complexity shortly. Clearly, DECOMPOSE(l, r, x_1, x_1) should be called once for each rule $(q, l) \xrightarrow{s} (q_1 \cdots q_k, r) \in R$. In lines 1–4 the structure of l and r is inspected and the properties $\mathrm{var}(l_i) = \mathrm{var}(l)$ and $\mathrm{var}(r_i) = \mathrm{var}(r)$ are tested in constant time. Mind that we precomputed $\mathrm{vars}(l)$ and $\mathrm{vars}(r)$, which can be done in linear time in the size of the rule. Then each subtree r_i is considered in lines 7–10 in constant time. Finally, we consider all direct input subtrees l_i in lines 12–18. The tests involving the variables are all performed in constant time due to the preprocessing step that computes $\mathrm{vars}(l)$ and $\mathrm{vars}(r)$. Moreover, at most one recursive call to DECOMPOSE is generated for each input subtree t_i. So if we implement the union in lines 18 and 19 by a constant-time operation (such as list concatenation, which can be done since it is trivially a disjoint union), then we obtain the linear time-complexity. □

6 Concluding remarks

In this paper we have shown how to restrict computations of WXTTs to given input and output WSA, and have discussed the relevance of this technique for parsing and translation applications over input strings, resulting in the computation of translation forests and other statistical parameters of interest. We have also shown how to factorize transducer rules, resulting in an asymptotic reduction in the complexity for these algorithms.

In machine translation applications transducers usually have very large sets of rules. One should then specialize the restriction construction in such a way that the number of useless rules for $\mathrm{Prod}(N_w, M)$ is considerably reduced, resulting in a more efficient construction. This can be achieved by grounding the construction of the new rules by means of specialized strategies, as usually done for parsing based on context-free grammars; see for instance the parsing algorithms by Younger (1967) or by Earley (1970).

References

André Arnold and Max Dauchet. 1975. *Transductions inversibles de forêts*. Thèse 3ème cycle M. Dauchet, Université de Lille.

André Arnold and Max Dauchet. 1976. Bi-transductions de forêts. In *ICALP*, pages 74–86. Edinburgh University Press.

André Arnold and Max Dauchet. 1982. Morphismes et bimorphismes d'arbres. *Theoret. Comput. Sci.*, 20(1):33–93.

Yehoshua Bar-Hillel, Micha Perles, and Eliyahu Shamir. 1964. On formal properties of simple phrase structure grammars. In Yehoshua Bar-Hillel, editor, *Language and Information: Selected Essays on their Theory and Application*, chapter 9, pages 116–150. Addison Wesley.

Jay Earley. 1970. An efficient context-free parsing algorithm. *Commun. ACM*, 13(2):94–102.

Samuel Eilenberg. 1974. *Automata, Languages, and Machines*, volume 59 of *Pure and Applied Math.* Academic Press.

Joost Engelfriet, Zoltán Fülöp, and Heiko Vogler. 2002. Bottom-up and top-down tree series transformations. *J. Autom. Lang. Combin.*, 7(1):11–70.

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*, EATCS Monographs on Theoret. Comput. Sci., chapter IX, pages 313–403. Springer.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule? In *Proc. HLT-NAACL*, pages 273–280. Association for Computational Linguistics.

Daniel Gildea, Giorgio Satta, and Hao Zhang. 2006. Factoring synchronous grammars by sorting. In *Proc. CoLing/ACL*, pages 279–286. Association for Computational Linguistics.

Jonathan Graehl and Kevin Knight. 2004. Training tree transducers. In *HLT-NAACL*, pages 105–112. Association for Computational Linguistics. See also (Graehl et al., 2008).

Jonathan Graehl, Kevin Knight, and Jonathan May. 2008. Training tree transducers. *Computational Linguistics*, 34(3):391–427.

Kevin Knight. 2007. Capturing practical natural language transformations. *Machine Translation*, 21(2):121–133.

Zhifei Li and Jason Eisner. 2009. First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proc. EMNLP*, pages 40–51. Association for Computational Linguistics.

Eric Lilin. 1981. Propriétés de clôture d'une extension de transducteurs d'arbres déterministes. In *CAAP*, volume 112 of LNCS, pages 280–289. Springer.

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

Mark-Jan Nederhof and Giorgio Satta. 2003. Probabilistic parsing as intersection. In *Proc. IWPT*, pages 137–148. Association for Computational Linguistics.

Rebecca Nesson, Giorgio Satta, and Stuart M. Shieber. 2008. Optimal k-arization of synchronous tree-adjoining grammar. In *Proc. ACL*, pages 604–612. Association for Computational Linguistics.

Jacques Sakarovitch. 2009. Rational and recognisable power series. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, EATCS Monographs on Theoret. Comput. Sci., chapter IV, pages 105–174. Springer.

Giorgio Satta and Enoch Peserico. 2005. Some computational complexity results for synchronous context-free grammars. In *Proc. HLT-EMNLP*, pages 803–810. Association for Computational Linguistics.

Marcel Paul Schützenberger. 1961. On the definition of a family of automata. *Information and Control*, 4(2–3):245–270.

Daniel H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Inform. Control*, 10(2):189–208.

Hao Zhang, Liang Huang, Daniel Gildea, and Kevin Knight. 2006. Synchronous binarization for machine translation. In *Proc. HLT-NAACL*, pages 256–263. Association for Computational Linguistics.