Tree Parsing with Synchronous Tree-Adjoining Grammars

Matthias Büchse [a] and **Mark-Jan Nederhof** [b] and **Heiko Vogler** [a]

[a] Department of Computer Science
Technische Universität Dresden
D-01062 Dresden, Germany

[b] School of Computer Science
University of St Andrews
North Haugh, St Andrews, KY16 9SX, UK

Abstract

Restricting the input or the output of a grammar-induced translation to a given set of trees plays an important role in statistical machine translation. The problem for practical systems is to find a compact (and in particular, finite) representation of said restriction. For the class of synchronous tree-adjoining grammars, partial solutions to this problem have been described, some being restricted to the unweighted case, some to the monolingual case. We introduce a formulation of this class of grammars which is effectively closed under input and output restrictions to regular tree languages, i.e., the restricted translations can again be represented by grammars. Moreover, we present an algorithm that constructs these grammars for input and output restriction, which is inspired by Earley's algorithm.

1 Introduction

Many recent systems for statistical machine translation (SMT) (Lopez, 2008) use some grammar at their core. Chiang (2007), e. g., uses synchronous context-free grammars (SCFG) that derive pairs of translationally equivalent sentences. Huang et al. (2006) use tree-to-string transducers (called xRLNS) that describe pairs of the form (phrase-structure tree, string). Other systems, such as (Eisner, 2003; Zhang et al., 2008; Nesson et al., 2006; DeNeefe and Knight, 2009), use variants of synchronous tree-adjoining grammars (STAGs) (Abeille et al., 1990; Joshi and Schabes, 1997) that derive pairs of dependency or phrase-structure trees. Common variants of STAGs are synchronous tree-substitution grammars (STSGs) and synchronous tree-insertion grammars (STIGs).

For grammar-based systems, a variety of tasks can be described using the general concepts of *input product* and *output product* (Maletti, 2010b). Roughly speaking, these products restrict the translation described by the grammar to a given tree or string language on the input or output side. For practical purposes, the derivations of the restricted translation are represented in a compact way, e.g., using a weighted regular tree grammar (WRTG) (Alexandrakis and Bozapalidis, 1987). The process of obtaining this representation is called *tree parsing* or *string parsing*, depending on the type of restriction. We illustrate the importance of input and output product by considering its role in three essential tasks of SMT.

Grammar Estimation. After the rules of the grammar have been obtained from a sample of translation pairs (rule extraction), the probabilities of the rules need to be determined. To this end, two approaches have been employed.

Some systems such as those by Chiang (2007) and DeNeefe and Knight (2009) hypothesize a canonical derivation for each translation pair, and apply relative-frequency estimation to the resulting derivations to obtain rule probabilities. While this procedure is computationally inexpensive, it only maximizes the likelihood of the training data under the assumption that the canonical derivations are the true ones.

Other systems such as those by Eisner (2003), Nesson et al. (2006), and Graehl et al. (2008) use a variant of the EM algorithm (Dempster et al., 1977) called Inside-Outside. This algorithm requires that the set of derivations for a given translation pair be representable by a WRTG. In most cases, this can be computed by restricting the grammar at hand to the given translation pair, that is, by applying input and output product. Note that the pair can contain strings or trees or even some combination thereof.

Feature Weight Estimation. In the systems mentioned at the beginning of this section, a probability distribution of the form $p(e, d \mid f)$ is chosen from a log-linear model (Berger et al., 1996; Och and Ney, 2002), where e, d, and f are an English

Proceedings of the 12th International Conference on Parsing Technologies, pages 14–25,
October 5-7, 2011, Dublin City University.

sentence, a derivation, and a foreign sentence, respectively. Such a distribution combines information from different sources, called features, such as the grammar or a probability distribution over English sentences. The features are represented by real-valued functions $h_i(e, d, f)$. For said combination, each feature gets a weight λ_i.

The feature weights are usually estimated using minimum-error-rate training (Och, 2003). For this it is necessary to compute, for a given f, the set D_f of n highest ranking derivations generating f on the foreign side. Roughly speaking, this set can be computed by applying the input product with f, and then applying the n-best algorithm (Huang and Chiang, 2005; Büchse et al., 2010). We note that, while f is usually a string, it can in some circumstances be a phrase-structure tree, as in (Huang et al., 2006).

Decoding. The actual translation, or decoding, problem amounts to finding, for a given f,

$$\hat{e} = \mathrm{argmax}_e \sum_d \prod_i h_i(e, d, f)^{\lambda_i} \ .$$

Even for the simplest grammars, this problem is NP hard (Casacuberta and de la Higuera, 2000). As a result, SMT systems use approximations such as crunching or variational decoding (Li et al., 2009). Here we focus on the former, which amounts to restricting the sum in the equation to the set D_f. Since this set is finite, the sum is then zero for almost all e, which makes the computation of $\hat{e}$ feasible. As mentioned before, the input product can be used to compute D_f.

As we have seen in these tasks, tree parsing is employed in recent SMT systems. Table 1 lists five relevant contributions in this area. These contributions can be classified according to a number of characteristics indicated by the column headings. One of these characteristics is the abstraction level (AL), which we categorize as follows:

1. language-theoretic result,
2. construction,
3. algorithm,
4. implementation.

The first three entries of Tab. 1 deal with contributions that are restricted to tree substitution. Huang et al. (2006) show an algorithm for computing the best derivation of the input product of an xRLNS with a single tree. Graehl et al. (2008) present an algorithm for computing the derivation WRTG for the input and output product of a tree-to-tree transducer (called xRLN) with a single pair of trees. Eisner (2003) describes an algorithm for computing the set of derivations for the input and output product of an STSG with a single pair of trees.

We note that the grammar classes covered so far are strictly less powerful than STAGs. This is due to the fact that STAGs additionally permit an operation called adjoining. As Nesson et al. (2006) and DeNeefe and Knight (2009) point out, the adjoining operation has a well-founded linguistic motivation, and permitting it improves translation quality.

There are two papers approaching the problem of tree parsing for STAGs, given in the fourth and fifth entries of the table. These papers establish closure properties, that is, their constructions yield a grammar of the same type as the original grammar. Since the resulting grammars are compact representations of the derivations of the input product or output product, respectively, these constructions constitute tree parsing.

Nederhof (2009) shows that weighted linear index grammars (WLIGs) are closed under weighted intersection with tree languages generated by WRTGs. WLIGs derive phrase-structure trees, and they are equivalent to tree-adjoining grammars (TAGs). His construction can be extended to some kind of synchronous WLIG without problems. However, synchronization interacts with the height restriction present for WLIG rules in a way that makes synchronous WLIGs less powerful than STAGs.

Maletti (2010a) uses an alternative representation of STAG, namely as extended tree transducers (XTT) with explicit substitution. In this framework, adjoining is encoded into the phrase-structure trees by introducing special symbols, to be evaluated in a separate step. He indicates that his representation of STAG is closed under input and output product with regular tree languages by providing a corresponding construction. However, in his setting, both the translations and the languages are unweighted.

The advantage of closure properties of the above kind is that they allow cascades of input and output products to be constructed in a uniform way, as well as applying further operations on the grammars, such as projection. Ultimately, SMT tasks may be described in this framework, as witnessed by toolboxes that exist for WFSTs (Mohri, 2009) and XTTs (May and Knight, 2006).

	characteristics			
paper	AL	grammar	restriction type	result
(Huang et al., 2006)	3–4	xRLNS	tree	best derivation
(Graehl et al., 2008)	3–4	xRLN	(tree, tree)	derivation WRTG
(Eisner, 2003)	3–4	STSG	(tree, tree)	derivations
(Nederhof, 2009)	2	WLIG	regular tree language	WLIG
(Maletti, 2010a)	2	XTT	regular tree language	XTT
(this paper)	1–3	WSTAG	regular tree language	WSTAG

Table 1: Tree-parsing algorithms published so far in comparison with this paper.

In this paper, we propose a weighted formulation of STAGs which is closed under input and output product with WRTGs, and we present a corresponding tree-parsing algorithm. This paper is organized as follows.

In Sec. 2, we introduce our formulation of STAGs, which is called weighted synchronous tree-adjoining grammar (WSTAG). The major difference with respect to the classical STAGs is twofold: (i) we use states and (ii) we encode substitution and adjoining sites as variables in the tree. The states make intersection with regular properties possible (without the need for relabeling as in (Shieber, 2004) and (Maletti, 2010a)). In addition, they permit implementing all features of conventional STAG/STIG, such as potential adjoining and left/right adjoining. The variables are used for synchronization of the input and output sides.

In Sec. 3, we show that WSTAGs are closed under input and output product with tree languages generated by WRTGs (cf. Theorem 1). We do this by means of a direct construction (cf. Sec. 4). Our construction is based on the standard technique for composing two top-down tree transducers (cf. page 195 of (Baker, 1979)). This technique has been extended in Theorem 4.12 of (Engelfriet and Vogler, 1985) to the composition of a macro tree transducer and a top-down tree transducer (also cf. (Rounds, 1970)); in fact, our direct construction is very similar to the latter one.

Section 5 contains Algorithm 1, which computes our construction (modulo reduction). It is inspired by a variant of Earley's algorithm (Earley, 1970; Graham et al., 1980). In this way we avoid computation of a certain portion of useless rules, and we ensure that the complexity is linear in the size of the input WSTAG. The algorithm is presented in the framework of deductive parsing (Goodman, 1999; Nederhof, 2003).

In Sections 6 and 7, we discuss the correctness of our algorithm and its complexity, respectively.

2 Formalisms

We denote the set of all unranked, ordered, labeled trees over some alphabet Σ by U_Σ. We represent trees as well-formed expressions, e.g., $\mathrm{S}(\mathrm{Adv}(\mathrm{yesterday}), *)$; a graphical representation of this tree occurs at the very bottom of Fig. 1(a). Sometimes we assign a *rank* (or: *arity*) $k \in \mathbb{N}$ to a symbol $\sigma \in \Sigma$ and then require that every σ-labeled position of a tree has exactly k successors. We denote the set of all positions of a tree $t \in U_\Sigma$ by $\mathrm{pos}(t)$. A position is represented as a finite sequence of natural numbers (Gorn notation). If $w \in \mathrm{pos}(t)$, then $t(w)$ denotes the label of t at w, and $\mathrm{rk}_t(w)$ denotes the number of successors of w.

2.1 Weighted Regular Tree Grammars

A *weighted regular tree grammar* (for short: WRTG) is a tuple $H = (P, \Sigma, r_0, R, p)$ where P is a finite set of *states*, Σ is an alphabet, $r_0 \in P$ is the *initial state*, and R is a finite set of *rules*; every rule ρ has the form $r \to \sigma(r_1, \ldots, r_k)$ where $k \in \mathbb{N}$, $r, r_1, \ldots, r_k \in P$, and $\sigma \in \Sigma$ (note that $\sigma(r_1, \ldots, r_k)$ is a tree over $\Sigma \cup P$); finally, $p : R \to \mathbb{R}_{\geq 0}$ is the *weight assignment*, where $\mathbb{R}_{\geq 0}$ is the set of all non-negative real numbers.

A *run (of H)* is a tree $\kappa \in U_P$. Let $t \in U_\Sigma$ and let κ be a run. We say that κ is a *run on t* if $\mathrm{pos}(\kappa) = \mathrm{pos}(t)$ and for every position $w \in \mathrm{pos}(t)$ the rule $\rho(\kappa, t, w)$ is in R, where $\rho(\kappa, t, w)$ is defined as

$$\kappa(w) \to t(w)\Big(\kappa(w1), \ldots, \kappa(w\,\mathrm{rk}_t(w))\Big).$$

The *weight of a run κ on t* is the value $\mathrm{wt}(\kappa, t) \in \mathbb{R}_{\geq 0}$ defined by

$$\mathrm{wt}(\kappa, t) = \prod_{w \in \mathrm{pos}(t)} p(\rho(\kappa, t, w)) \ .$$

The *weighted tree language generated by* WRTG H is the mapping $L(H) : U_\Sigma \to \mathbb{R}_{\geq 0}$

defined by

$$L(H)(t) = \sum_{\substack{\kappa \text{ run on } t \\ \kappa(\varepsilon)=r_0}} \mathrm{wt}(\kappa, t) \ .$$

The *support* $\mathrm{supp}(L(H))$ of $L(H)$ is the set of all $t \in U_\Sigma$ such that $L(H)(t) \neq 0$.

2.2 Weighted Tree-Adjoining Grammars

In our formulation of STAGs we use states (cf. Fülöp et al. (2010) for the use of states in STSGs). The states make intersection with devices of finite-state power possible. More specifically, they allow for a product construction as is common in automata theory (cf. Sec. 4). Moreover, they permit implementing all features of conventional STAG/STIG, such as potential adjoining and left/right adjoining. In contrast to the traditional setting, substitution sites and adjoining sites are formalized as explicit positions in the right-hand sides of rules; these positions are labeled by variables. For this, we assume that $x_1, x_2, \ldots$ and $z_1, z_2, \ldots$ are two fixed infinite and disjoint sequences of pairwise distinct variables. We assume that every x_j has rank 0 and every z_j has rank 1. We use $x_1, x_2, \ldots$ to denote substitution sites and $z_1, z_2, \ldots$ to denote adjoining sites. The foot node is labeled by an additional nullary symbol $*$. The input and output components are synchronized via these sites and the corresponding states.

A *weighted synchronous tree-adjoining grammar with states* (for short: WSTAG) is a tuple $G = (Q, F, \Sigma, q_0, R, p)$ where

- Q and F are disjoint finite sets (of *nullary* and *unary states*, respectively, each denoted by variants of q and f, respectively),
- Σ is an alphabet (*terminal alphabet*),
- q_0 is a nullary state (*initial state*),
- R is a finite set of *rules* of either of the following forms:

$$q \rightarrow \langle \zeta\zeta', q_1 \cdots q_m, f_1 \cdots f_l \rangle \qquad (\alpha)$$
$$f(*) \rightarrow \langle \zeta\zeta', q_1 \cdots q_m, f_1 \cdots f_l \rangle \qquad (\beta)$$

  where ζ and ζ' are trees over $\Sigma \cup V$ and

$$V = \{x_1, \ldots, x_m, z_1, \ldots, z_l\} \qquad (\alpha)$$
$$V = \{x_1, \ldots, x_m, z_1, \ldots, z_l, *\} \qquad (\beta)$$

  and every element of V occurs exactly once in each of ζ and ζ', and
- $p : R \rightarrow \mathbb{R}_{\geq 0}$ is the *weight assignment*.

Rules of the forms (α) and (β) are called (m, l)-rules; ζ and ζ' are called the *input tree* and the *output tree* of the rule, respectively. For fixed q and f, the sets of all rules of the form (α) and (β) are denoted by R_q and R_f, resp. Figure 1(a) shows an example of a WSTAG. In the following, let $G = (Q, F, \Sigma, q_0, R, p)$ be a WSTAG.

We define the semantics in terms of bimorphisms (Shieber, 2006). For this we define a WRTG $\mathcal{H}_G$ that generates the weighted tree language of derivation trees of G, and two tree homomorphisms h_1 and h_2 that retrieve from a derivation tree the derived input tree and output tree, respectively.

The *derivation tree WRTG of G* is the WRTG $\mathcal{H}_G = (Q \cup F, R, q_0, R', p')$ where

- we assign the rank $m + l$ to every (m, l)-rule,
- R' is the set of all rules $D(\rho)$ with $\rho \in R$ and
  - if ρ is of the form (α), then $D(\rho) = q \rightarrow \rho(q_1, \ldots, q_m, f_1, \ldots, f_l)$,
  - if ρ is of the form (β), then $D(\rho) = f \rightarrow \rho(q_1, \ldots, q_m, f_1, \ldots, f_l)$,
- and $p'(D(\rho)) = p(\rho)$.

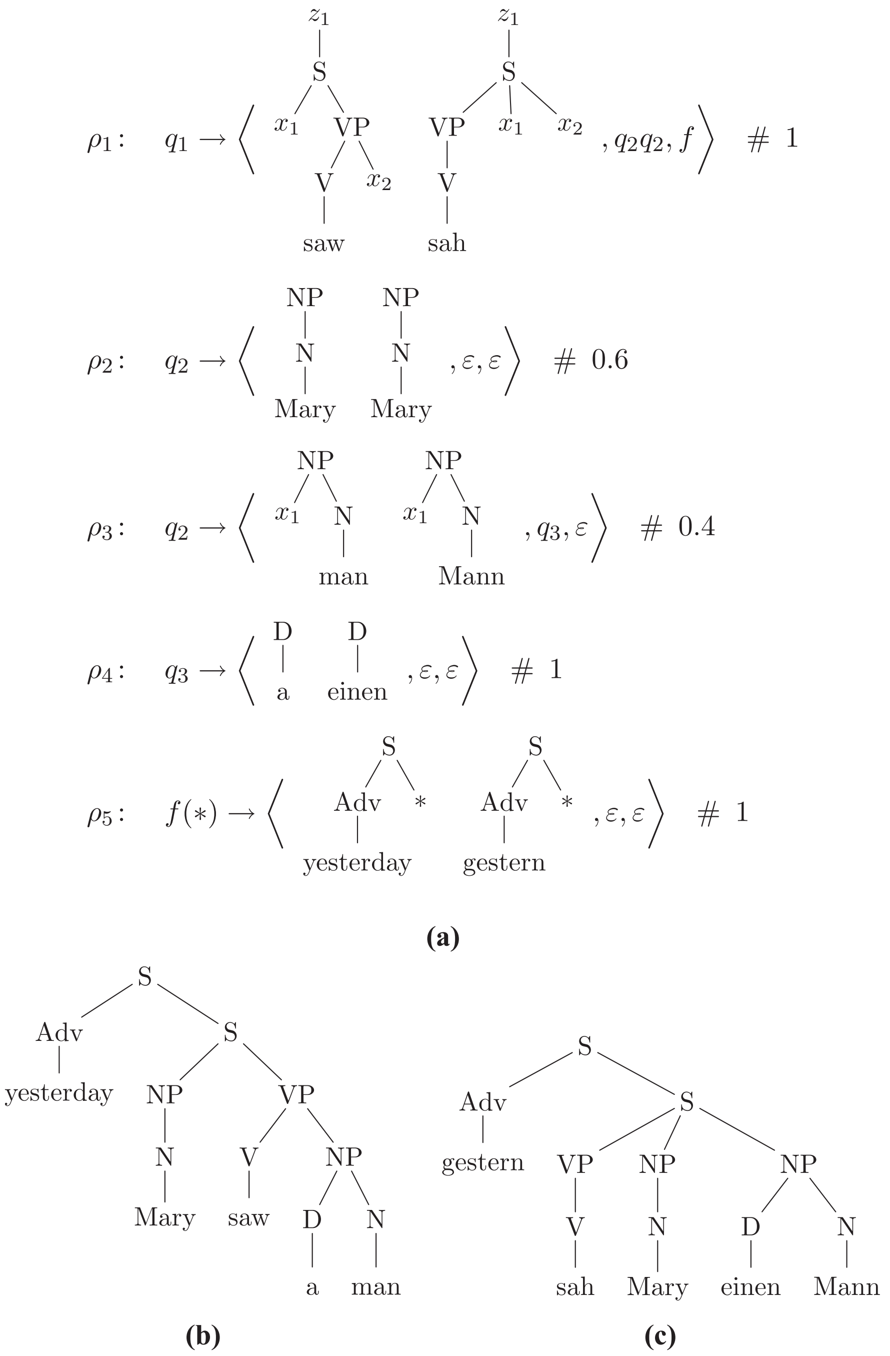
Recall that $L(\mathcal{H}_G)$ is the weighted tree language generated by $\mathcal{H}_G$. We call the trees in $\mathrm{supp}(L(\mathcal{H}_G))$ *derivation trees*. For instance, with the WSTAG of Fig. 1(a), the tree $d_{\mathrm{ex}} = \rho_1(\rho_2, \rho_3(\rho_4), \rho_5)$ is a derivation tree of the derived trees shown in Fig. 1(b) and Fig. 1(c).

Formally, the derived trees are obtained by the tree homomorphisms h_1 and h_2, each of type $\mathrm{supp}(L(\mathcal{H}_G)) \rightarrow U_\Sigma$, which we define inductively as follows:

$$h_1(\rho(d_1, \ldots, d_m, d'_1, \ldots, d'_l))$$
$$= \zeta[\vec{x}/h_1(\vec{d})][\![\vec{z}/h_1(\vec{d'})]\!]$$

where $[\vec{x}/h_1(\vec{d})]$ abbreviates the m first-order substitutions $[x_1/h_1(d_1)] \ldots [x_m/h_1(d_m)]$, and $[\![\vec{z}/h_1(\vec{d'})]\!]$ abbreviates the l second-order substitutions $[\![z_1/h_1(d'_1)]\!] \ldots [\![z_l/h_1(d'_l)]\!]$. The first-order substitution $[x/s]$ replaces every occurrence of x by s, and the second-order substitution $[\![z/s]\!]$ is defined inductively in Fig. 2. The tree homomorphism h_2 is defined in the same way as h_1, but with ζ' instead of ζ. We call d a derivation tree of the pair $(h_1(d), h_2(d))$.

In continuation of our running example, we calculate $h_1(d_{\mathrm{ex}})$, where we use [1], [2], and [![3]!] as abbreviations for the substitutions $[x_1/h_1(\rho_2)]$,


ρ1: q1 → ⟨ z1 S x1 VP V saw x2 , z1 S VP V sah x1 x2 , q2q2, f ⟩ # 1
ρ2: q2 → ⟨ NP N Mary , NP N Mary , ε, ε ⟩ # 0.6
ρ3: q2 → ⟨ NP x1 N man , NP x1 N Mann , q3, ε ⟩ # 0.4
ρ4: q3 → ⟨ D a , D einen , ε, ε ⟩ # 1
ρ5: f(∗) → ⟨ S Adv yesterday ∗ , S Adv gestern ∗ , ε, ε ⟩ # 1
S Adv yesterday S NP N Mary VP V saw NP D a N man
S Adv gestern S VP V sah NP N Mary NP D einen N Mann

(a)

(b)

(c)

Figure 1: (a) Example of a WSTAG (following (Joshi and Schabes, 1997)), (b) input tree, and (c) output tree.

$$\sigma(\zeta_1, \dots, \zeta_k)[\![z/s]\!] = \sigma(\zeta_1[\![z/s]\!], \dots, \zeta_k[\![z/s]\!]) \qquad x_j[\![z/s]\!] = x_j \qquad *[\![z/s]\!] = *$$

$$z_j(\zeta)[\![z/s]\!] = \begin{cases} s' & \text{if } z = z_j \text{ and } s' \text{ is obtained from } s \text{ by replacing the } * \text{ by } \zeta \\ z_j(\zeta[\![z/s]\!]) & \text{otherwise.} \end{cases}$$

Figure 2: Second-order substitution $[\![z/s]\!]$, where $\sigma \in \Sigma$, x_j is a nullary variable, and z_j is a unary variable.

$[x_2/h_1(\rho_3(\rho_4))]$, and $[\![z_1/h_1(\rho_5)]\!]$, respectively:

$$\begin{aligned} & h_1(d_{\text{ex}}) \\ & = z_1(\mathrm{S}(x_1, \mathrm{VP}(\mathrm{V}(\mathrm{saw}), x_2)))[1][2][\![3]\!] \\ & = z_1(\mathrm{S}(h_1(\rho_2), \mathrm{VP}(\mathrm{V}(\mathrm{saw}), x_2)))[2][\![3]\!] \\ & = z_1(\underbrace{\mathrm{S}(h_1(\rho_2), \mathrm{VP}(\mathrm{V}(\mathrm{saw}), h_1(\rho_3(\rho_4)))))}_{\xi}[\![3]\!] \\ & = \mathrm{S}(\mathrm{Adv}(\mathrm{yesterday}), \xi) \ , \end{aligned}$$

which evaluates to the tree of Fig. 1(b).

The *weighted tree transformation specified by WSTAG G* is $T(G) : U_\Sigma \times U_\Sigma \to \mathbb{R}_{\geq 0}$ defined by

$$T(G)(s,t) = \sum_{d \text{ derivation tree of } (s,t)} L(\mathcal{H}_G)(d).$$

We note that for every derivation tree d there is a unique run κ_d of $\mathcal{H}_G$ on d: $\kappa_d(w)$ is the state occurring in the left-hand side of the rule $d(w)$, for every $w \in \mathrm{pos}(d)$.

As an example, we reconsider d_{ex}, which is a derivation tree of the translation pair (s,t), given by Fig. 1(b) and Fig. 1(c). Then we have $L(\mathcal{H}_G)(d_{\text{ex}}) = \mathrm{wt}(\kappa_{d_{\text{ex}}}, d_{\text{ex}}) = \prod_{i=1}^{5} p(\rho_i) = 0.24$. Since d_{ex} is the only derivation tree of (s,t), we have that $T(G)(s,t) = 0.24$.

STSGs as defined in Fülöp et al. (2010) are WSTAGs which only have nullary states. Also classical STAGs (Abeille et al., 1990; Shieber and Schabes, 1990) and STIGs (Nesson et al., 2005; Nesson et al., 2006; DeNeefe and Knight, 2009) can be viewed as particular WSTAGs. In particular, potential adjoining as it occurs in classical STAGs can be simulated, as the following excerpt of a WSTAG shall illustrate:

$$\begin{aligned} q &\to \langle z_1(A(x_1))\, \zeta', q', f\rangle \\ f(*) &\to \langle *\, *, \varepsilon, \varepsilon\rangle \\ f(*) &\to \langle z_1(*)\, z_1(*), \varepsilon, f'\rangle \end{aligned}$$

Moreover, the left-/right adjoining restriction of STIGs can be handled by keeping appropriate finite information in the states.

3 Closure Result

First we define the input product and output product of a weighted tree transformation $T \colon U_\Sigma \times U_\Sigma \to \mathbb{R}_{\geq 0}$ and a weighted tree language $L \colon U_\Sigma \to \mathbb{R}_{\geq 0}$. Formally, the *input product of T and L* is the weighted tree transformation $L \lhd T \colon U_\Sigma \times U_\Sigma \to \mathbb{R}_{\geq 0}$ such that

$$(L \lhd T)(s,t) = L(s) \cdot T(s,t) \ .$$

Similarly, we define the *output product of T and L* as $T \rhd L \colon U_\Sigma \times U_\Sigma \to \mathbb{R}_{\geq 0}$ such that

$$(T \rhd L)(s,t) = T(s,t) \cdot L(t) \ .$$

We note that the input product and output product can be considered as a kind of composition of weighted relations by viewing L as mapping $L' : U_\Sigma \times U_\Sigma \to \mathbb{R}_{\geq 0}$ with $L'(s,t) = L(s)$ if $s = t$, and $L'(s,t) = 0$ otherwise.

Here we prove that WSTAGs are closed under input product (and output product) with WRTGs.

Theorem 1 *For every WSTAG G and WRTG H there are WSTAGs $H \lhd G$ and $G \rhd H$ such that*

- *$T(H \lhd G) = L(H) \lhd T(G)$ and*
- *$T(G \rhd H) = T(G) \rhd L(H)$.*

We will only prove the closure under input product, because the proof for the output product is similar.

For the unweighted case, the closure result follows from classical results. The unweighted case is obtained if we replace the algebra in Sec. 2 by another one: $\mathbb{R}_{\geq 0}$ is replaced by the set $\mathbb{B} = \{\mathit{true}, \mathit{false}\}$ and the operations $+$ and $\cdot$ are replaced by disjunction and conjunction, respectively. In other words, we replace the inside semiring $(\mathbb{R}_{\geq 0}, +, \cdot, 0, 1)$ by the Boolean semiring $(\mathbb{B}, \vee, \wedge, \mathit{false}, \mathit{true})$. Then $L(H)$ and $T(G)$ become sets $L(H) \subseteq U_\Sigma$ and $T(G) \subseteq U_\Sigma \times U_\Sigma$. In this setting and using $h : \mathrm{supp}(L(\mathcal{H}_G)) \to U_\Sigma \times U_\Sigma$ defined by $h(d) = (h_1(d), h_2(d))$, we have that

$$L(H) \lhd T(G) = h(h_1^{-1}(L(H))) \ .$$

Note that $h_1^{-1}(L(H)) \subseteq \mathrm{supp}(L(\mathcal{H}_G))$. Now we observe that h_1 can be computed by a particular macro tree transducer (Engelfriet, 1980; Courcelle and Franchi-Zannettacci, 1982); we note that in (Shieber, 2006) such macro tree transducers were called embedded tree transducers.

Engelfriet and Vogler (1985) proved in Theorem 7.4 that the class of regular tree languages is closed under the inverse of macro tree transducers. Thus, since $L(H)$ is a regular tree language, also $h_1^{-1}(L(H))$ is a regular tree language. Thus $L(H) \lhd T(G) = h(L(\bar{H}))$ for some regular tree grammar $\bar{H}$. Now it is easy to construct (using h_1 and h_2) a STAG $H \lhd G$ from $\bar{H}$ such that $T(H \lhd G) = h(L(\bar{H}))$.

For the weighted case, we give a direct construction in the next section.

4 Direct Construction

We now provide our construction of the WSTAG $H \lhd G$ in Theorem 1 where $G = (Q, F, \Sigma, q_0, R, p)$ is a WSTAG and $H = (P, \Sigma, r_0, R_H, p_H)$ is a WRTG.

First we define an enrichment of H that can generate trees like ζ as they occur in rules of G, that is, including variables x_j, z_j, and possibly $*$. To this end, let $\rho \in R$. A *(state) assignment for* ρ is a mapping θ that maps each nullary variable in ρ to a state of H and each unary variable in ρ to a pair of such states. Likewise, if $*$ occurs in ζ, then θ maps it to a state of H. For every $r \in P$ and assignment θ, we define the WRTG $H(r, \theta)$, which is obtained from H by using r as initial state and adding the following rules with weight 1 for every $r, r' \in P$:

$$\begin{aligned} & r \to x_j && \text{if } \theta(x_j) = r, \\ & r \to z_j(r') && \text{if } \theta(z_j) = (r, r'), \text{ and} \\ & r \to * && \text{if } \theta(*) = r\ . \end{aligned}$$

Roughly speaking, for every rule $\rho \in R$, we let $H(r, \theta)$ "run" on the input tree ζ of ρ. Formally, we define the *product WSTAG of* H *and* G as the WSTAG $H \lhd G =$

$$(Q \times P, F \times (P \times P), \Sigma, (q_0, r_0), R', p')$$

as follows. Let $\rho \in R$, $r \in P$, and θ an assignment for ρ. Then, depending on whether ρ has the form (α) or (β), the rule

$$(q, r) \to \langle \zeta\zeta', u, v \rangle \qquad (\alpha)$$

$$(f, (r, \theta(*))) \to \langle \zeta\zeta', u, v \rangle \qquad (\beta)$$

is in R' where

- $u = (q_1, \theta(x_1)) \cdots (q_m, \theta(x_m))$ and
- $v = (f_1, \theta(z_1)) \cdots (f_m, \theta(z_l))$.

We denote this rule by (ρ, r, θ). Its weight is $p'(\rho, r, \theta) = p(\rho) \cdot L(H(r, \theta))(\zeta)$. There are no further elements in R'.

We omit a proof for $T(H \lhd G) = L(H) \lhd T(G)$.

We have that $|R'| \in O(|R| \cdot |P|^C)$ where $C = \max\{m + 2 \cdot l + y \mid \exists\rho\colon \rho \text{ is an } (m, l)\text{-rule}, y = 1 \text{ in case } (\alpha), y = 2 \text{ in case } (\beta)\}$.

5 Algorithm

Now we present Algorithm 1, which performs the construction of $H \lhd G$. It uses a strategy similar to that of Earley's algorithm to construct at least all useful rules of $H \lhd G$ while avoiding construction of a certain portion of useless rules. A rule is *useful* if it occurs in some derivation tree; otherwise it is *useless*.

Algorithm 1 Product construction algorithm

```
Require: $G = (Q, F, \Sigma, q_0, R, p)$ a WSTAG and
         $H = (P, \Sigma, r_0, R_H, p_H)$ a WRTG,
Ensure: $R_u$ contains at least the useful rules of $H \lhd G$,
    $p_u$ coincides with the weight assignment of $H \lhd G$
    on $R_u$
                                         ▷ step 1: compute $\mathcal{I}$
$\mathcal{I} \leftarrow \emptyset$
repeat
    add items to $\mathcal{I}$ by applying the rules in Fig. 3
until convergence
                                         ▷ step 2: compute rules
$R_u \leftarrow \emptyset$
for $[\rho, \varepsilon, r, \theta] \in \mathcal{I}$ do
    $R_u \leftarrow R_u \cup \{(\rho, r, \theta)\}$ as in Sec. 4
                                         ▷ step 3 (optional): reduce
perform reachability analysis to remove useless
    rules from $R_u$
                                         ▷ step 4: compute weights
for $(\rho, r, \theta) \in R_u$ do
    $p_u(\rho, r, \theta) \leftarrow p(\rho) \cdot \mathcal{W}([\rho, \varepsilon, r, \theta])$
                                         ▷ defined in Fig. 4
```

Conceptually, the algorithm proceeds in four steps. Note that, in practice, some of these steps may be implemented interleaved in order to reduce constants in the runtime complexity.

The first step is based on a deductive system, or deductive parsing schema, which is given in Fig. 3. Its central notion is that of an *item*, which is a syntactic representation of a proposition. We say that an item *holds* if the corresponding proposition is true. In Sec. 6 we will explain the meaning of the items in detail. Roughly speaking, the items drive a depth-first left-to-right simulation of H on the trees on the input side of rules of G. Items with round brackets are responsible for top-down traversal and items with square brackets for horizontal and bottom-up traversal. The deductive system contains inference rules which are, as usual, syntactic representations of conditional implications (Goodman, 1999; Nederhof, 2003).

The first step of the algorithm computes the least set $\mathcal{I}$ of items that is closed under application of the inference rules. This is done in the usual iterative way, starting with the empty set and applying rules until convergence. Since there are only finitely many items, this process will terminate. Note that, given the soundness of the infer-

$$\text{(1)} \quad \frac{}{(q_0, r_0)}$$

$$\text{(2q)} \quad \frac{(q, r)}{(\rho, \varepsilon, r)} \ \{\ \rho \in R_q \qquad\qquad \text{(2f)} \quad \frac{(f, r)}{(\rho, \varepsilon, r)} \ \{\ \rho \in R_f$$

$$\text{(3q)} \quad \frac{(q, r) \qquad [\rho, \varepsilon, r, \theta]}{[q, r]} \ \{\ \rho \in R_q \qquad \text{(3f)} \quad \frac{(f, r) \qquad [\rho, \varepsilon, r, \theta]}{[f, r, \theta(*)]} \ \{\ \rho \in R_f$$

$$\text{(4)} \quad \frac{(\rho, w, r)}{[\rho, w, 0, r, r_1 \cdots r_k, \emptyset]} \ \left\{ \begin{array}{c} \zeta(w) \in \Sigma \quad k = \mathrm{rk}_\zeta(w) \\ p_H(r \to \zeta(w)(r_1, \ldots, r_k)) > 0 \end{array} \right.$$

$$\text{(5)} \quad \frac{[\rho, w, j, r, r_1 \cdots r_k, \theta]}{(\rho, w(j+1), r_{j+1})} \ \{\ \zeta(w) \in \Sigma \quad k = \mathrm{rk}_\zeta(w) \quad 1 \leq j+1 \leq k$$

$$\text{(6)} \quad \frac{[\rho, w, j, r, r_1 \cdots r_k, \theta] \qquad [\rho, w(j+1), r_{j+1}, \theta']}{[\rho, w, j+1, r, r_1 \cdots r_k, \theta \cup \theta']} \ \{\ \zeta(w) \in \Sigma \quad k = \mathrm{rk}_\zeta(w)$$

$$\text{(7)} \quad \frac{[\rho, w, k, r, r_1 \cdots r_k, \theta]}{[\rho, w, r, \theta]} \ \{\ \zeta(w) \in \Sigma \quad k = \mathrm{rk}_\zeta(w)$$

$$\text{(8x)} \quad \frac{(\rho, w, r)}{(q_j, r)} \ \{\ \zeta(w) = x_j \qquad\qquad \text{(8z)} \quad \frac{(\rho, w, r)}{(f_j, r)} \ \{\ \zeta(w) = z_j$$

$$\text{(9x)} \quad \frac{(\rho, w, r) \qquad [q_j, r]}{[\rho, w, r, \{x_j \mapsto r\}]} \ \{\ \zeta(w) = x_j \quad \text{(9z)} \quad \frac{(\rho, w, r) \qquad [f_j, r, r']}{(\rho, w1, r')} \ \{\ \zeta(w) = z_j$$

$$\text{(10)} \quad \frac{(\rho, w, r) \qquad [f_j, r, r'] \qquad [\rho, w1, r', \theta]}{[\rho, w, r, \theta \cup \{z_j \mapsto (r, r')\}]} \ \{\ \zeta(w) = z_j$$

$$\text{(11)} \quad \frac{(\rho, w, r)}{[\rho, w, r, \{* \mapsto r\}]} \ \{\ \zeta(w) = *$$

Note: whenever ρ is mentioned, we implicitly assume that
$\rho = q \to \langle \zeta\zeta', q_1 \cdots q_m, f_1 \cdots f_l \rangle$ or
$\rho = f(y) \to \langle \zeta\zeta', q_1 \cdots q_m, f_1 \cdots f_l \rangle$.

Figure 3: Deductive parsing schema for the input product.

If $\zeta(w) \in \Sigma$, with $k = \mathrm{rk}_\zeta(w)$, and θ_j is the restriction of θ to variables below node wj in ρ, then:

$$\mathcal{W}([\rho, w, r, \theta]) = \sum_{\substack{r_1, \ldots, r_k : \\ [\rho, w1, r_1, \theta_1], \ldots, [\rho, wk, r_k, \theta_k] \in \mathcal{I}}} p_H(r \to \zeta(w)(r_1, \ldots, r_k)) \cdot \prod_j \mathcal{W}([\rho, wj, r_j, \theta_j])$$

If $\zeta(w) \in \{*, x_1, \ldots, x_m\}$, then $\mathcal{W}([\rho, w, r, \theta]) = 1$.

If $\zeta(w) = \{z_j\}$, with $\theta(z_j) = (r, r')$, then $\mathcal{W}([\rho, w, r, \theta]) = \mathcal{W}([\rho, w1, r', \theta \setminus \{z_j \mapsto (r, r')\}])$

Figure 4: Computing the weights of rules in the product grammar $H \lhd G$.

ence rules (cf. Sec. 6), all items in $\mathcal{I}$ hold.

In the second step, for each $[\rho, \varepsilon, r, \theta]$ in $\mathcal{I}$ we construct one rule of $H \lhd G$ as in (α) or as in (β) in Sec. 4, depending on whether $\rho \in R_q$ or $\rho \in R_f$.

The third step is a reachability analysis to remove useless rules. This step is optional—it depends on the application whether the runtime spent here is amortized by subsequent savings.

In the fourth step, we determine the weight of each of the rules we have. For a rule (ρ, r, θ) this is $p(\rho) \cdot \mathcal{W}([\rho, \varepsilon, r, \theta])$, where $\mathcal{W}$ is defined in Fig. 4. The computation can be sped up by storing "back pointers" for each item, i.e., the items which were used for its generation. Alternatively, it is possible to compute the weights on-the-fly during the first step, thus alleviating the need for a separate recursive computation.

To this end, items should be prioritized to make sure that they are generated in the right order for the computation. To be more precise, one has to ensure that all items referred to on the right-hand sides of the equations in Fig. 4 are generated before the items on the left-hand sides.

6 Meaning of Items

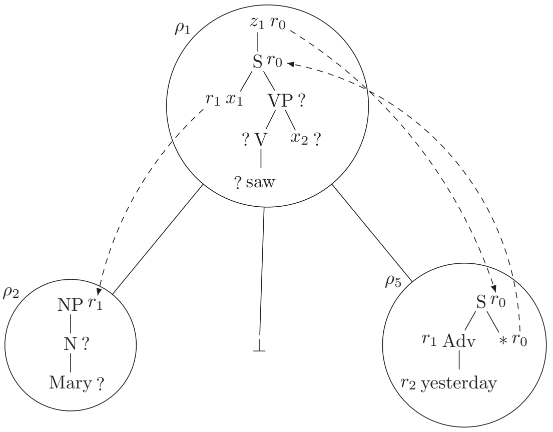
The meaning of the items can best be illustrated by the concepts of enriched derivation tree and partial enriched derivation tree.

An *enriched derivation tree* is a modified derivation tree in which the labels have the form (ρ, c) for some rule ρ and some decoration mapping c; c maps every position of the input tree ζ of ρ to a state of the WRTG H. Moreover, c must be consistent with the rules of H, and positions that coincide in the derived tree must be decorated with the same state (cf. Fig. 5, dashed lines). A *partial enriched derivation tree* (for short: pedt) is an enriched derivation tree in which subtrees can still be missing (represented by $\bot$) or the decoration with states from H is not yet complete (i.e., some positions are mapped to ?).

Figure 5 shows an example pedt d, where we represent the decoration at each position of d by annotating the corresponding input tree. This pedt can be viewed as representing application of the following rules:

$$(1), (2q), (8z), (2f), (4), (5), (4), (5), (4), (7), \ldots$$

Now we make our description of pedts more precise. Let n be a position of d, ρ be the rule occurring in the label $d(n)$, and ζ be the input tree of ρ. Then there is a position w of ζ such that (1) every position u which is lexicographically smaller than w is decorated by a state and, if $\zeta(u)$ is a variable x_j or z_j, then the subtree of n which corresponds to the variable does not contain $\bot$ or ?, and (2) every position v which is lexicographically greater than w is decorated by ? and, if $\zeta(v)$ is a variable x_j or z_j, then the child of n which corresponds to the variable is $\bot$. For instance, if we consider the root $n = \varepsilon$ of the pedt in Fig. 5, then $w = 11$ (i.e., the position with label x_1).



$$
\begin{aligned}
R_H: \quad & r_0 \to \mathrm{S}(r_1, r_3) \\
& r_1 \to \mathrm{Adv}(r_2) \\
& r_2 \to \mathrm{yesterday} \\
& \vdots
\end{aligned}
$$

Figure 5: Partial enriched derivation tree.

Finally we can describe the meaning of the items by referring to properties of pedts.

$\boldsymbol{(q, r)}$ (and $\boldsymbol{(f, r)}$): There are a pedt d, a position n of d, a rule ρ, and a decoration c such that $d(n) = (\rho, c)$, $\rho \in R_q$ (resp., $\rho \in R_f$), and $c(\varepsilon) = r$.

$\boldsymbol{[q, r]}$: There are a pedt d, a position n of d, a rule ρ, and a decoration c such that $d(n) = (\rho, c)$, $\rho \in R_q$, $c(\varepsilon) = r$, and c is a complete decoration.

$\boldsymbol{[f, r, r']}$: There are a pedt d, a position n of d, a rule ρ, and a decoration c such that $d(n) = (\rho, c)$, $\rho \in R_f$, $c(\varepsilon) = r$, c is a complete decoration, and c maps the $*$-labeled position of the input tree of ρ to r'.

$\boldsymbol{(\rho, w, r)}$: There are a pedt d, a position n of d, and a decoration c such that $d(n) = (\rho, c)$ and $c(w) = r$.

$\boldsymbol{[\rho, w, r, \theta]}$: There are a pedt d, a position n of d, and a decoration c such that $d(n) = (\rho, c)$,

$c(w) = r$ and, if a position u below w of the input tree ζ of ρ is labeled by x_j, then $c(u) = \theta(x_j)$, and if it is labeled by z_j, then $(c(u), c(u1)) = \theta(z_j)$, and if it is labeled by $*$, then $c(u) = \theta(*)$.

$[\boldsymbol{\rho, w, j, r, r_1 \cdots r_k, \theta}]$: There are a pedt d, a position n of d, and a decoration c such that $d(n) = (\rho, c)$, $c(w) = r$, and, if a position u of the input tree ζ of ρ is labeled by some variable y and u is lexicographically smaller than wj, c and θ agree in the same way as in the preceding item.

Given this semantics of items, it is not difficult to see that the inference rules of the deduction system are sound. The completeness of the system can be derived by means of a small proof by contradiction.

7 Complexity Analysis

In this section, we analyse the worst-case space and time complexity of step 1 of Algorithm 1.

The space complexity is

$$O\big(|G|_{\text{in}} \cdot |R_H| \cdot |P|^C\big) ,$$

which is determined by the number of possible items of the form $[\rho, w, j, r, r_1 \cdots r_k, \theta]$. The first factor, $|G|_{\text{in}}$, denotes the *input size of* G, defined by $\sum_{\rho \in R} |\text{pos}(\zeta(\rho))|$, where $\zeta(\rho)$ is the input tree of ρ. It captures the components ρ, w, and j in said items, which together identify exactly one node of an input tree of G. The factor $|R_H|$ captures the components r and $r_1 \cdots r_k$. The final factor, $|P|^C$, captures the θ, where C is given at the end of Sec. 4.

Following McAllester (2002) we determine the time complexity by the number of instantiations of the inference rules. In our case the time complexity coincides with the space complexity.

8 Conclusion

We have introduced a formulation of STAGs that is closed under input product and output product with regular weighted tree languages. By the result of Maletti and Satta (2009), this implies closure under input product and output product with regular weighted string languages. We have provided a direct construction of the STAG that generates said input product (and, mutatis mutandis, the output product). No such construction has been published before that deals with both weights and synchronization. Moreover, we have presented a novel algorithm for computing our construction. This algorithm is inspired by Earley's algorithm to the effect that computation of a certain portion of useless rules is avoided.

The next step towards an implementation would be to consider pruning. This amounts to partitioning the set $\mathcal{I}$ of items and imposing a bound on the size of each partition. Such a technique has already been presented by Chiang (2007) for his cube-pruning algorithm.

Another possible future contribution could be an algorithm specifically tailored to the input product with a regular weighted string language. For this scenario, several contributions exist, requiring additional restictions however. For instance, Nesson et al. (2006) show a CYK-like algorithm for intersecting a STIG with a pair of strings. Their algorithm requires that the trees of the grammar be binarized. As DeNeefe and Knight (2009) point out, this makes the grammar strictly less powerful. They in turn propose a construction which converts the STIG into an equivalent tree-to-string transducer, and they use corresponding algorithms for parsing, such as the one by DeNero et al. (2009). However, their construction relies on the fact that tree-insertion grammars are weakly equivalent to context-free grammars. Thus, it is not applicable to the more general STAGs.

Acknowledgments

We are grateful to the referees for thorough remarks and helpful suggestions. The first author was financially supported by DFG VO 1011/6-1.

References

A. Abeille, Y. Schabes, A.K. Joshi. 1990. Using lexicalized TAGs for machine translation. In *Proc. of COLING*, vol. 3, pp. 1–6, Helsinki, Finland.

A. Alexandrakis, S. Bozapalidis. 1987. Weighted grammars and Kleene's theorem. *Inform. Process. Letters*, 24(1):1–4.

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

A.L. Berger, V.J. Della Pietra, S.A. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Comp. Ling.*, 22:39–71.

M. Büchse, D. Geisler, T. Stüber, H. Vogler. 2010. n-best parsing revisited. In: F. Drewes,

M. Kuhlmann (eds.), *Proc. of Workshop on ATANLP*, pp. 46–54. ACL.

F. Casacuberta, C. de la Higuera. 2000. Computational complexity of problems on probabilistic grammars and transducers. In: A.L. Oliveira (ed.), *Proc. of ICGI*, LNAI 1891, pp. 15-24, Springer-Verlag.

D. Chiang. 2007. Hierarchical phrase-based translation. *Comp. Ling.*, 33(2):201–228.

B. Courcelle, P. Franchi-Zannettacci. 1982. Attribute grammars and recursive program schemes I and II. Theoret. Comput. Sci. 17, 163-191 and 235-257.

A. P. Dempster, N. M. Laird, D. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *J. of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38.

S. DeNeefe, K. Knight. 2009. Synchronous tree adjoining machine translation. In: P. Koehn, R. Mihalcea (eds.), *Proc. of EMNLP*, pp. 727–736. ACL.

S. DeNeefe, K. Knight, H. Vogler. 2010. A decoder for probabilistic synchronous tree insertion grammars. In: F. Drewes, M. Kuhlmann (eds.), *Proc. of Workshop on ATANLP*, pp. 10–18. ACL.

J. DeNero, M. Bansal, A. Pauls, and D. Klein. 2009. Efficient parsing for transducer grammars. In *Proc. NAACL*, pp. 227–235.

J. Earley. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.

J. Eisner. 2003. Learning non-isomorphic tree mappings for machine translation. In *Proc. of 41st ACL - Volume 2*, ACL, pages 205–208, Stroudsburg, PA, USA. Association for Computational Linguistics.

J. Engelfriet. 1980. Some open questions and recent results on tree transducers and tree languages. In: R.V. Book (ed.), *Formal language theory; perspectives and open problems*, New York, Academic Press, 241-286.

J. Engelfriet, H. Vogler. 1985. Macro tree transducers. J. Comput. System Sci. 31, 71–146.

Z. Fülöp, A. Maletti, H. Vogler. 2010. Preservation of recognizability for synchronous tree substitution grammars. In: F. Drewes, M. Kuhlmann (eds.), *Proc. of Workshop on ATANLP*, pp. 1–9. ACL.

M. Galley, M. Hopkins, K. Knight, D. Marcu. 2004. What's in a translation rule? In: D. Marcu, S. Dumais, S. Roukos (eds.), *Proc. of HLT-NAACL*, pp. 273–280, ACL.

F. Gécseg, M. Steinby. 1997. Tree languages. In: G. Rozenberg, A. Salomaa (eds.), *Handbook of Formal Languages*, vol. 3, chapter 1, pp. 1–68. Springer-Verlag.

J. Goodman. 1999. Semiring parsing. *Comput. Linguist.*, 25(4):573–605.

J. Graehl, K. Knight, J. May. 2008. Training tree transducers. *Comput. Linguist.*, 34(3):391–427.

S. L. Graham, M. Harrison, and W. L. Ruzzo. 1980. An improved context-free recognizer. *ACM Trans. Program. Lang. Syst.*, 2:415–462, July.

L. Huang, D. Chiang. 2005. Better k-best parsing. In *Proc. of IWPT*, pp. 53–64, ACL.

L. Huang, K. Knight, and A. Joshi. 2006. Statistical syntax-directed translation with extended domain of locality. In *Proc. AMTA*, pages 66–73.

A.K. Joshi, L.S. Levy, M. Takahashi. 1975. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1):136–163.

A.K. Joshi, Y. Schabes. 1997. Tree-adjoining grammars. In: *Handbook of Formal Languages*. Chapter 2, pp. 69–123, Springer-Verlag.

K. Knight. 2007. Capturing practical natural language transformations. Machine Translation 21, 121–133.

Z. Li, J. Eisner, S. Khudanpur. 2009. Variational decoding for statistical machine translation. In: *Proc. of ACL-IJCNLP*, pp. 593–601, ACL.

A. Lopez. 2008. Statistical Machine Translation. ACM Computing Surveys 40(3), 8:1–8:49.

A. Maletti. 2010a. A tree transducer model for synchronous tree-adjoining grammars. In J.-S. Chang and P. Koehn, eds., *Proc. ACL*, pp. 1067–1076. ACL.

A. Maletti. 2010b. Input and output products for weighted extended top-down tree transducers. In Y. Gao, H. Lu, S. Seki, and S. Yu, eds., *Proc. DLT*, LNCS, vol. 6224, pp. 316–327. Springer-Verlag.

A. Maletti and G. Satta. 2009. Parsing algorithms based on tree automata. In: *Proc. IWPT*, pp. 1–12. ACL.

D.F. Martin and S.A. Vere. 1970. On syntax-directed transduction and tree transducers. In: *Proc. 2nd ann. Assoc. Comput. Sci. Symp. Theory of Comp.*, pp. 129–135.

J. May and K. Knight. 2006. Tiburon: a weighted tree automata toolkit. In O.H. Ibarra and H.-C. Yen, editors, *CIAA 2006*, volume 4094 of *Lecture Notes in Comput. Sci.*, pages 102–113. Springer-Verlag.

D. McAllester. 2002. On the complexity analysis of static analyses. *J. ACM*, 49:512–537, July.

M. Mohri. 2009. Weighted automata algorithms. In M. Droste, W. Kuich, and H. Vogler, editors, *Handbook of Weighted Automata*, chapter 6, pages 213–254. Springer-Verlag.

M.-J. Nederhof. 2003. Weighted deductive parsing and Knuth's algorithm. *Computational Linguistics*, 29(1):135–143.

M.-J. Nederhof. 2009. Weighted parsing of trees. In: *Proc. of IWPT*, pp. 13–24. ACL.

R. Nesson, S.M. Shieber, A. Rush. 2005. Induction of probabilistic synchronous tree-insertion grammars. Technical Report TR-20-05, Computer Science Group, Harvard University, Cambridge, Massachusetts.

R. Nesson, S.M. Shieber, A. Rush. Induction of probabilistic synchronous tree-inserting grammars for machine translation. In: *Proc. of the 7th AMTA*, 2006.

F.J. Och. 2003. Minimum error rate training in statistical machine translation. In: *Proc. of Annual Meeting of ACL*, pp. 160–167.

F.J. Och, H. Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. In: *Proc. of Annual Meeting of ACL*, pp. 295–302. ACL.

W.C. Rounds. 1969. Context-free grammars on trees. In: *Proc. of STOC*, pp. 143–148.

W.C. Rounds. 1970. Tree-oriented proofs of some theorems on context-free and indexed languages In: *2nd Ann. ACM STOC*, pp. 109–116.

S.M. Shieber. 2004. Synchronous grammars and tree transducers. In: *Proc. 7th Workshop on Tree Adjoining Grammars and Related Formalisms*, pp. 88–95. ACL.

S.M. Shieber. 2006. Unifying synchronous tree-adjoining grammars and tree transducers via bimorphisms. In: *Proc. of EACL*, pp. 377–384. ACL.

S.M. Shieber, Y. Schabes. 1990. Synchronous tree-adjoining grammars. In: *Proc. of COLING*, vol. 3, pp. 253–258, ACL.

M. Zhang, H. Jiang, A. Aw, H. Li, C.L. Tan, S. Li. 2008. A tree sequence alignment-based tree-to-tree translation model. In: J.D. Moore, S. Teufel, J. Allan, S. Furui, *Proc. of Annual Meeting of ACL*, pp. 559–567. ACL.