State-Split for Hypergraphs with an Application to Tree-Adjoining Grammars

Johannes Osterholzer and **Torsten Stüber**
Department of Computer Science
Technische Universität Dresden
D-01062 Dresden
{johannes.osterholzer,torsten.stueber}@tu-dresden.de

Abstract

In this work, we present a generalization of the state-split method to probabilistic hypergraphs. We show how to represent the derivational stucture of probabilistic tree-adjoining grammars by hypergraphs and detail how the generalized state-split procedure can be applied to such representations, yielding a state-split procedure for tree-adjoining grammars.

1 Introduction

The *state-split* method (Petrov et al., 2006) allows the successive refinement of a probabilistic context-free grammar (PCFG) for the purpose of natural language processing (NLP). It employs automatic subcategorization of nonterminal symbols: in an iteration of a *split-merge cycle,* every nonterminal of the PCFG is split into two, along with the corresponding grammar rules, whose probabilities are distributed uniformly to the split rules. The resulting PCFG's rule probabilities are then trained on an underlying treebank using the *Expectation-Maximization* algorithm (Dempster et al., 1977) for maximum-likelihood estimation on incomplete data. Finally, split nonterminal symbols which do not contribute to a significant increase in likelihood are merged back together. This counteracts an exponential blowup in the number of nonterminals and prevents, to some degree, the phenomenon of overfitting.

The in-house developed statistical machine translation toolkit *Vanda* (Büchse et al., 2012) offers state-splitting for the refinement of its language models. Vanda's internal representation of the various weighted tree grammar, automaton and transducer formalisms utilized for translation is by means of probabilistic *hypergraphs,* i.e., graphs consisting of *vertices* and *hyperedges*, where each of the latter connects a (possibly empty) sequence of *tail vertices* to a *head vertex* and is assigned a probability. Hence, our implementation of state-split operates on such hypergraphs as underlying data structures. The connection between parsing and hypergraphs is well-known in the field of NLP (Klein and Manning, 2004), where a hypergraph represents the derivation forest of a certain word. In our system, however, we use such a probabilistic hypergraph to represent the *whole* derivational structure of a grammar, with a one-to-one correspondence between hyperpaths and grammar derivations. We apply well-known product constructions from the theory of weighted automata, going back to Bar-Hillel et al. (1961) and generalized to the case of weighted tree automata by Maletti and Satta (2009), to restrict them to the derivations of a given word.

The mildly context-sensitive generative capacity of *tree-adjoining grammars (TAG)* is well-suited to the purpose of NLP (Joshi and Schabes, 1991). tree-adjoining grammars allow two basic operations to rewrite and derive trees: *substitution,* where a tree's leaf node is replaced with another tree, and *adjoining,* which can be seen as the second-order substitution of a context (called an auxiliary tree) into another tree.

Probabilistic tree-adjoining grammars (PTAG) (Schabes, 1992; Resnik, 1992) assign to every derived tree an associated probability. They can also be incorporated into Vanda by an appropriate representation with hypergraphs. Such PTAGs can be extracted from treebank corpora, as detailed by Chen et al. (2006). The idea to refine these grammars furthermore by executing the already implemented state-split procedure on their hypergraph representations arises naturally. The main contribution of the work at hand is a formalization of this

Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+11), pages 180–188,
Paris, September 2012.

idea. However, one should note that our proposed method does not just apply to PTAG, but should carry over to many grammar formalisms that can be represented by hypergraphs. In our formalization, we have to deal with a complication, introduced by the nature of adjunction in TAG. For this purpose, we introduce *split relations.*

An alternative to our approach to state-splitting PTAG is the one taken by Shindo et al. (2012), where a method for symbol refinement of probabilistic tree substitution grammars is presented. Since tree substitution grammars are just tree-adjoining grammars without adjoining sites (and indeed, adjoining can be simulated by the combination of a tree substitution grammar which encodes adjoining explicitly, and a yield function which performs these encoded operations, cf. (Maletti, 2010)), it is possible that their technique can be adapted to the more general setting. This would have the additional advantage that smoothing by backoff to simpler context-free grammar rules is incorporated in their system, increasing performance in the case of sparse data, while we do not cover smoothing in the work at hand.

Note that, although most of the preliminaries may be safely skimmed, we want to point out that our definitions of first- and second-order substitution are slightly non-standard (but they enable a concise formalization of TAG).

2 Preliminaries

In the following, we will denote the set of non-negative integers by $\mathbb{N}$. The set $\{1, \dots, k\}$ shall be abbreviated by $[k]$. The set of the non-negative real numbers will be denoted by $\mathbb{R}_{\geq 0}$, and the closed interval between two reals a and b by $[a, b]$.

The set of finite words over a set A is written as A^*, $\varepsilon \in A^*$ is the empty word and $A^+ = A^* \setminus \{\varepsilon\}$. The reflexive-transitive closure of a binary relation R shall be denoted by R^*.

2.1 Unranked Trees

We call a finite set Σ of *symbols* an *alphabet.* Given a finite set A and alphabet Σ, let Σ_A denote the alphabet of all σ_a with $\sigma \in \Sigma$, $a \in A$. Note that the new symbol σ_a is merely a syntactic construct and should be identified neither with σ nor a.

Presuming an alphabet Σ and set A, the set $U_\Sigma(A)$ of *unranked trees over* Σ *indexed by* A is the smallest set U such that $A \subseteq U$ and for every $n \in \mathbb{N}$, $t_1, \dots, t_n \in U$, and $\sigma \in \Sigma$, also $\sigma(t_1, \dots, t_n) \in U$. For a tree $\sigma()$ we will just write σ and $U_\Sigma(\emptyset)$ will be denoted by U_Σ. The set of *positions* $\operatorname{pos}(t)$ of a tree $t \in U_\Sigma(A)$ is defined by $\operatorname{pos}(\sigma(t_1, \dots, t_n)) = \{\varepsilon\} \cup \{iw \mid i \in [n], w \in \operatorname{pos}(t_i)\}$ and $t(w)$ denotes the *label* of t at position w.

Throughout the rest of the paper, let $X = \{x_1, x_2, \dots\}$ denote an infinite set of variables and, for $n \in \mathbb{N}$, let $X_n = \{x_1, \dots, x_n\}$. Similarly, let $Y = \{y_1, y_2, \dots\}$, let $Y_n = \{y_1, \dots, y_n\}$ for $n \in \mathbb{N}$, and $Z = \{z\}$. Given $k \in \mathbb{N}$, we call a tree $t \in U_\Sigma(A \cup \Delta_X)$ (resp. $t \in U_{\Sigma \cup \Delta_Y}(A)$) *proper in* X_k *(resp.* Y_k*)* if for every $i > k$, there is no appearance of δ_{x_i} (resp. of δ_{y_i}) in t, and for every $i \in [k]$, there is exactly one position in t labeled with δ_{x_i} (resp. δ_{y_j}), for some $\delta \in \Delta$. This unique $\delta \in \Delta$ will be denoted by $\operatorname{lb}_t(x_i)$ (resp. $\operatorname{lb}_t(y_j)$).

For an alphabet Σ and $k \in \mathbb{N}$, we denote first-order substitution of the trees $s_1, \dots, s_k \in U_\Sigma$ into $t \in U_\Sigma(\Sigma_{X_k})$ by $t[x_1/s_1, \dots, x_k/s_k] \in U_\Sigma$, abbreviated by $t[\boldsymbol{x}/\boldsymbol{s}]$. The tree $t[\boldsymbol{x}/\boldsymbol{s}]$ is the result of replacing every node in t which is labeled by σ_{x_i}, for $i \in [k]$ and some $\sigma \in \Sigma$, by the tree s_i. Similarly, for $s \in U_\Sigma(A)$ and $t \in U_\Sigma(Z)$, let $t[z/s]$ denote the tree obtained from t by replacing every node that is labelled with z by s.

For every alphabet Σ and $k \in \mathbb{N}$, second-order substitution of the trees $s_1, \dots, s_k \in U_\Sigma(Z)$ into the tree $t \in U_{\Sigma \cup \Sigma_{Y_k}}$ will be denoted by $t[\![y_1/s_1, \dots, y_k/s_k]\!] \in U_\Sigma$, or shorter, by $t[\![\boldsymbol{y}/\boldsymbol{s}]\!]$. For every symbol $\sigma \in \Sigma$ and variable $y_i \in Y_k$, we define $\sigma_{y_i}(t_1, \dots, t_n)[\![\boldsymbol{y}/\boldsymbol{s}]\!] = s_i[z/\sigma(t_1[\![\boldsymbol{y}/\boldsymbol{s}]\!], \dots, t_n[\![\boldsymbol{y}/\boldsymbol{s}]\!])]$, and we define $\sigma(t_1, \dots, t_n)[\![\boldsymbol{y}/\boldsymbol{s}]\!] = \sigma(t_1[\![\boldsymbol{y}/\boldsymbol{s}]\!], \dots, t_n[\![\boldsymbol{y}/\boldsymbol{s}]\!])$ for $\sigma \in \Sigma$.

2.2 Hypergraphs

A *hypergraph* is a tuple (V, E, μ, g) where the set V contains the graph's *vertices* and E the *hyperedges* (or just *edges*). Edge connectivity is denoted by the function $\mu \colon E \to V^+$ and $g \in V$ is the graph's *goal vertex.* For a hyperedge e with $\mu(e) = a_0 a_1 \cdots a_n$, we define its *head* as $\operatorname{hd}(e) = a_0$, its *tail* as $\operatorname{tl}(e) = a_1 \cdots a_n$ and its *arity* as $\operatorname{ar}(e) = n$. The set of a*-hyperpaths* H_G^a of a hypergraph $G = (V, E, \mu, g)$, $a \in V$, is the largest set of trees $H \subseteq U_E$ such that for every $d \in H$, $w \in \operatorname{pos}(d)$, we have $\operatorname{hd}(d(\varepsilon)) = a$, $|\{v \in \operatorname{pos}(d) \mid v = wi, i \in \mathbb{N}\}| = |\operatorname{tl}(e)|$, and $\operatorname{tl}(d(w)) = \operatorname{hd}(d(w1)) \cdots \operatorname{hd}(d(wn))$, where

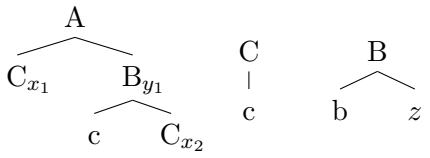

Figure 1: PTAG $\mathcal{G}$ with trees α_1, α_2 and β

$n = |\,\mathrm{tl}(e)|$. The *hyperpaths* of G are its g-hyperpaths: $H_G = H_G^g$.

A tuple (V, E, μ, g, p) is called a *probabilistic hypergraph (phg)* if (V, E, μ, g) is a hypergraph and $p\colon E \to [0, 1]$ assigns a *probability* to every hyperedge. We will denote the class of all probabilistic hypergraphs by $\mathcal{H}$ and assume definitions for unweighted hypergraphs to carry over to the probabilistic case. The *probability* of a derivation $d \in H_G$ is defined to be

$$P(d \mid G) = \prod_{w \in \mathrm{pos}(d)} p(d(w)).$$

In the following, we will fix the phg G (resp. G', G'') to be of the form (V, E, μ, g, p) (resp. (V', E', μ', g', p'), etc.).

2.3 Tree-Adjoining Grammars

In our definition of tree-adjoining grammars, the variable x_i (resp. y_j) in a symbol σ_{x_i} (resp. σ_{y_j}) will be used to tag σ with the information that it labels the ith substitution (resp. jth adjoining) site of the respective tree. The variable z in an auxiliary tree denotes the tree's foot node. This formalization allows us, among others, to give a straightforward definition of derived trees.

A *tree-adjoining grammar (TAG)* is a tuple $\mathcal{G} = (\Sigma, S, \mathcal{S}, \mathcal{A})$ where Σ is an alphabet, $S \in \Sigma$ is called the *start symbol*, $\mathcal{S} \subseteq U_\Gamma(\Delta)$ the set of *initial trees*, and $\mathcal{A} \subseteq U_\Gamma(\Delta \cup Z)$ the set of *auxiliary trees*, with $\Gamma = \Sigma \cup \Sigma_Y$ and $\Delta = \Sigma_X$. We demand for every $t \in \mathcal{S} \cup \mathcal{A}$ that $t(\varepsilon) \in \Sigma$, and that there are $n, m \in \mathbb{N}$ such that t is proper in X_n and in Y_m. In the following, we will denote these unique n and m by $\mathrm{rk}_1(t)$, resp. $\mathrm{rk}_2(t)$. Moreover, we require that for every $t \in \mathcal{A}$, z appears exactly once in t.

A *probabilistic tree-adjoining grammar (PTAG)* is then a tuple $\mathcal{G} = (\Sigma, S, \mathcal{S}, \mathcal{A}, P, Q)$ such that $(\Sigma, S, \mathcal{S}, \mathcal{A})$ is a TAG, $P\colon \mathcal{S} \cup \mathcal{A} \to [0, 1]$ maps trees to their probabilities and $Q = (Q_t\colon Y_{\mathrm{rk}_2(t)} \to [0, 1])_{t \in \mathcal{S} \cup \mathcal{A}}$ assigns to adjoining sites the probability of their activation. We will denote the class of all PTAG by $\mathcal{T}$.

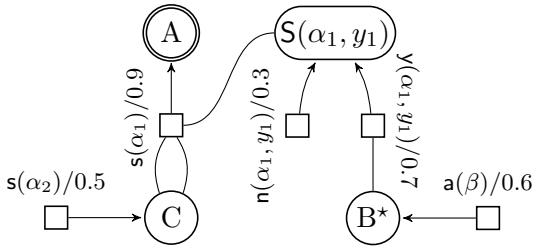

Figure 2: Hypergraph representation of $\mathcal{G}$

In the following, the PTAG $\mathcal{G}$ (resp. $\mathcal{G}'$) will be assumed to be of the form $(\Sigma, S, \mathcal{S}, \mathcal{A}, P, Q)$ (resp. $(\Sigma', S', \mathcal{S}', \mathcal{A}', P', Q')$).

Let us examine an example PTAG $\mathcal{G}$, with $\Sigma = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{c}, \mathrm{b}\}$, $S = \mathrm{A}$, initial trees $\mathcal{S} = \{\alpha_1, \alpha_2\}$, and an auxiliary tree $\mathcal{A} = \{\beta\}$. Refer to Fig. 1, which depicts the tree α_1 with two substitution sites labeled, resp., with symbols A and C, as well as one adjoining site, labeled with B. To the right are the initial tree α_2 and the auxiliary tree β, both with no substitution or adjoining sites. The foot node of β is indicated by z.

We denote the conditional probability that an elementary tree t with root symbol A is used to rewrite substitution or adjoining sites that are labeled with A by $P(t)$. For $\mathcal{G}$, e.g., let $P(\alpha_1) = 0.9$, $P(\alpha_2) = 0.5$, and $P(\beta) = 0.6$. The probability that the adjoining site in α_1, tagged with y_1, is activated during a derivation, is denoted by $Q_{\alpha_1}(y_1)$. In the case of this running example, let $Q_{\alpha_1}(y_1) = 0.7$.

3 Hypergraph Representations

For every PTAG $\mathcal{G}$, we can construct a hypergraph $G = \mathrm{hg}(\mathcal{G})$, whose hyperpaths stand in a one-to-one correspondence to the grammar's derivations. Hence, we will call G a *hypergraph representation* of $\mathcal{G}$.

In our following definition, such a hypergraph $\mathrm{hg}(\mathcal{G})$ can contain three different types of vertices. The vertices of the first type model the derivation of initial trees. These vertices are just copies of the symbols of $\mathcal{G}$: for every symbol $A \in \Sigma$, a vertex A is introduced. Refer to Fig. 2, which depicts the hypergraph $\mathrm{hg}(\mathcal{G})$ that represents the derivational structure of the PTAG $\mathcal{G}$ from our running example. Its A- and C-hyperpaths model derivation of initial

trees with respective root symbols. Other, irrelevant, vertices of this form are omitted from the figure.

The second vertex type helps in the derivation of auxiliary trees. For each $A \in \Sigma$, the hypergraph contains a corresponding vertex $A^\star$. In Fig. 2, the only relevant vertex of this form is $\mathrm{B}^\star$, whose hyperpaths represent derivations of auxiliary trees with root symbol B.

Lastly, for every adjoining site that appears in an elementary tree t and is tagged with y_i, we include a vertex $\mathsf{S}(t, y_i)$. This vertex is used to explicitly model the decision to activate or not to activate the corresponding adjoining site. In the figure, there is only one such vertex, $\mathsf{S}(\alpha_1, y_1)$.

The hyperedges of $\mathrm{hg}(\mathcal{G})$ can also be classified by their intended meaning: a hyperedge of the form $\mathsf{s}(\alpha)$ signifies the substitution of the initial tree α during a derivation. Its head vertex is its root symbol. Its tail vertices are, in order, the labels of α's substitution sites (indicating that the derivation must continue with derivations of trees with the respective symbols at their root) and the vertices $\mathsf{S}(\alpha, y_i)$ modelling its adjoining sites (these indicate the necessary decision on activating the sites). In Fig. 2, we see, among others, $\mathsf{s}(\alpha_1)$, whose head vertex A corresponds to α_1's root symbol, while its two tail vertices C stand for its respective substitution sites, and $\mathsf{S}(\alpha_1, y_1)$ for its adjoining site.

Hyperedges of the form $\mathsf{a}(\beta)$, with β an elementary auxiliary tree, possess essentially the same structure as the former, but signify derivation of auxiliary trees. In the figure, the only edge of this form is $\mathsf{a}(\beta)$. Its head vertex is $\mathrm{B}^\star$ because β's root symbol is B, and it has no tail vertices, since there are no sites in β.

Finally, the hyperedge $\mathsf{y}(t, y_i)$ (resp. $\mathsf{n}(t, y_i)$) encodes the information that an adjoining site in t was activated (deactivated). Both have $\mathsf{S}(t, y_i)$ as head, and while $\mathsf{n}(t, y_i)$ has no tail, the tail of $\mathsf{y}(t, y_i)$ signifies that the derivation should continue with the adjoining of an auxiliary tree. In Fig. 2, e.g., $\mathsf{y}(\alpha_1, y_1)$ models the activation of the site that corresponds to its head vertex $\mathsf{S}(\alpha_1, y_1)$, and has the tail vertex $\mathrm{B}^\star$, because that site is labeled with B.

The probabilities of these hyperedges are taken over from P and Q in the obvious way. Formally, we define $\mathrm{hg}(\mathcal{G}) = G$ with

$$\begin{aligned}
V =& \{A, A^\star \mid A \in \Sigma\} \\
&\cup \{\mathsf{S}(t, y_i) \mid t \in \mathcal{S} \cup \mathcal{A}, i \in [\mathrm{rk}_2(t)]\}, \\
E =& \{\mathsf{s}(t) \mid t \in \mathcal{S}\} \cup \{\mathsf{a}(t) \mid t \in \mathcal{A}\} \\
&\cup \{\mathsf{y}(t, y_i), \mathsf{n}(t, y_i) \mid t \in \mathcal{S} \cup A, i \in [\mathrm{rk}_2(t)]\},
\end{aligned}$$

goal vertex $g = S$, and

$$\begin{aligned}
\mu(\mathsf{s}(t)) &= A\, B_1 \cdots B_n\, \mathsf{S}(t, y_1) \cdots \mathsf{S}(t, y_m), \\
\mu(\mathsf{a}(t)) &= A^\star\, B_1 \cdots B_n\, \mathsf{S}(t, y_1) \cdots \mathsf{S}(t, y_m), \\
\mu(\mathsf{y}(t, y_j)) &= \mathsf{S}(t, y_j)\, C_j^\star, \\
\mu(\mathsf{n}(t, y_j)) &= \mathsf{S}(t, y_j),
\end{aligned}$$

where $n = \mathrm{rk}_1(t)$, $m = \mathrm{rk}_2(t)$, $A = t(\varepsilon)$, $B_i = \mathrm{lb}_t(x_i)$ for $i \in [n]$, and $C_j = \mathrm{lb}_t(y_j)$ for $j \in [m]$, while

$$\begin{aligned}
p(\mathsf{s}(t)) &= P(t), \quad p(\mathsf{y}(t, y_j)) = Q_t(y_j), \\
p(\mathsf{a}(t)) &= P(t), \quad p(\mathsf{n}(t, y_j)) = 1 - Q_t(y_j).
\end{aligned}$$

Note that for our simple running example from Fig. 1, there are only two possible derivations of trees with root symbol A: in the first one, we substitute two instances of α_2 into α_1, and, after activation of the adjoining site in α_1, adjoin β into this site. The corresponding A-hyperpath in Fig. 2 is $d_1 = \mathsf{s}(\alpha_1)\big(\mathsf{s}(\alpha_2), \mathsf{s}(\alpha_2), \mathsf{y}(\alpha_1, y_1)\big(\mathsf{a}(\beta)\big)\big)$. Alternatively, we can ignore the adjoining site, and arrive at the corresponding derivation $d_2 = \mathsf{s}(\alpha_1)\big(\mathsf{s}(\alpha_2), \mathsf{s}(\alpha_2), \mathsf{n}(\alpha_1, y_1)\big)$.

Given such a hyperpath that represents a PTAG derivation, we can compute the derivation's derived tree in a bottom-up manner with the function $\mathrm{yd}\colon H_{\mathrm{hg}(\mathcal{G})} \to U_\Sigma(X \cup Z)$ defined by

$$\begin{aligned}
&\mathrm{yd}(\mathsf{s}(t)(d_1, \ldots, d_n, d'_1, \ldots, d'_m)) \\
&\quad = t[\boldsymbol{x}/\mathrm{yd}(\boldsymbol{d})][\![\boldsymbol{y}/\mathrm{yd}(\boldsymbol{d'})]\!] \\
&\mathrm{yd}(\mathsf{a}(t)(d_1, \ldots, d_n, d'_1, \ldots, d'_m)) \\
&\quad = t[\boldsymbol{x}/\mathrm{yd}(\boldsymbol{d})][\![\boldsymbol{y}/\mathrm{yd}(\boldsymbol{d'})]\!] \\
&\mathrm{yd}(\mathsf{y}(t, y_j)(d)) = \mathrm{yd}(d) \\
&\mathrm{yd}(\mathsf{n}(t, y_j)) = z,
\end{aligned}$$

where again $n = \mathrm{rk}_1(t)$, $m = \mathrm{rk}_2(t)$, and $\mathrm{yd}(\boldsymbol{d})$ denotes the element-wise application of yd to all $d_1, \ldots, d_n$, analogously for $\mathrm{yd}(\boldsymbol{d'})$.

For example, for d_1 from above, we can compute its derived tree as

$$\begin{aligned}
\mathrm{yd}(d_1) &= \alpha_1[x_1/\alpha_2, x_2/\alpha_2][\![y_1/\beta]\!] \\
&= \mathrm{A}\big(\mathrm{C}(\mathrm{c}), \mathrm{B}_{y_1}(\mathrm{c}, \mathrm{C}(\mathrm{c}))\big)[\![y_1/\beta]\!] \\
&= \mathrm{A}\big(\mathrm{C}(\mathrm{c}), \beta\big[z/\mathrm{B}(\mathrm{c}, \mathrm{C}(\mathrm{c}))\big]\big) \\
&= \mathrm{A}\big(\mathrm{C}(\mathrm{c}), \mathrm{B}\big(\mathrm{b}, \mathrm{B}(\mathrm{c}, \mathrm{C}(\mathrm{c}))\big)\big).
\end{aligned}$$

4 State-Split Hypergraphs

In this section, we will detail how to generalize the state-split method presented by Petrov et al. to hypergraphs that represent PTAGs.

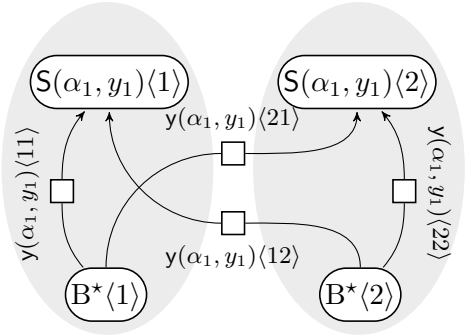

Figure 3: Crossing hyperedges

As explained in the introduction, the state-split algorithm proceeds in three distinct phases: first of all, it splits every vertex into two, and then it trains the probabilities of the resulting hyperedges on a given treebank corpus, using the EM algorithm. In a third step, vertices which do not increase likelihood are merged back together. However, due to the nature of adjunction in tree-adjoining grammar, the splits and merges cannot be quite as liberal as in the case of context-free grammars.

The reason for this will be shown immediately, but first and foremost, let us define how to represent split vertices. Given a hypergraph G, for any $a \in V$, and $b \in \{0, 1, 2\}$, define

$$a\langle b\rangle = \begin{cases} (a, b) & \text{if } b \in \{1, 2\} \\ a & \text{if } b = 0. \end{cases}$$

For a vertex a, the so-defined $a\langle 1\rangle$ and $a\langle 2\rangle$ will denote the two vertices which result from splitting a, while $a\langle 0\rangle$ is just a notation for the merged-back-together vertex a. We will also have to annotate split hyperedges in this way: for a hyperedge $e \in E$ with $\mathrm{ar}(e) = n$, as well as a tuple $\boldsymbol{b} \in \{0, 1, 2\}^{n+1}$, we introduce the syntax $e\langle \boldsymbol{b}\rangle$, which stands for $(e, \boldsymbol{b})$.

Given this notation, we can examine the mentioned complication of the state-split procedure, which arises when splitting hyperedges of the form $\mathsf{y}(t, y_j)$. Let us assume that the vertices $\mathsf{S}(\alpha_1, y_1)$, $\mathrm{B}^\star$, as well as the hyperedge $\mathsf{y}(\alpha_1, y_1)$ from Fig. 2 are to undergo subcategorization.

If we split the vertices and hyperedges indiscriminately, we arrive at the situation displayed in Fig. 3: each of the vertices has been split into two copies, and four new y-hyperedges were introduced. What is the meaning of these hyperedges? In a hyperpath representing a PTAG derivation, the appearance of the hyperedge $\mathsf{y}(\alpha_1, y_1)\langle 11\rangle$ signifies that the adjoining site in α_1, labeled with the split symbol $\mathrm{B}\langle 1\rangle$, is activated, and next, an auxiliary tree with equal root symbol must be adjoined into it.

However, there is a problem with the hyperedge $\mathsf{y}(\alpha_1, y_1)\langle 21\rangle$ which crosses the shaded ellipses in the figure (and, analogously, with $\mathsf{y}(\alpha_1, y_1)\langle 12\rangle$). This hyperedge can be interpreted as activation of the mentioned adjoining site, labeled with $\mathrm{B}\langle 1\rangle$, and preparation of adjoining an auxiliary tree with root symbol $\mathrm{B}\langle 2\rangle$. This stands in conflict to the concept of adjoining, where the label of the node to be replaced must be identical to the symbol at the root of the auxiliary tree.

There are several distinct possibilities to handle this complication. First of all, one could just do away with the above condition regarding adjoining. Actually, such a relaxation of the formalism was already proposed by Rogers (2003), resulting in *non-strict* tree-adjoining grammars.

As a second option, the formalism of TAG could be extended by introducing states (cf. (Büchse et al., 2011) for synchronous TAG). In this modification, the derivational structure of the grammar is no longer dependent on the labels of adjoining and substitution sites, this information is instead encoded into the states, which only appear as intermediate symbols in a derivation. Performing the splits and merges on such states, instead of symbols, could also remedy the problem.

However, for this work, we chose to stick to a conceptually simpler solution, thus staying close to the established notion of tree-adjoining grammar: we just disallow the creation of such crossing hyperedges during the split-merge cycle; or, to put it differently, we only introduce a hyperedge $\mathsf{y}(t, y)\langle i\, j\rangle$ into the split hypergraph if $i = j$. Hence, in Fig. 3, only the two hyperedges $\mathsf{y}(\alpha_1, y_1)\langle 11\rangle$ and $\mathsf{y}(\alpha_1, y_1)\langle 22\rangle$, which are both "within" the two shaded ellipses, would be generated.

To formalize this idea, we augment the state-split method with what we call *split relations*. Given a hypergraph G which is to undergo a split-merge cycle, a split relation on G is a symmetric relation $R \subseteq V \times V$ on the graph's vertices. If a hyperedge e from G connects, among others, two vertices a_1 and a_2 such that $a_1 R a_2$, then the idea is to split e only into such hyperedges which connect either only $a_1\langle 1\rangle$ and $a_2\langle 1\rangle$, or only $a_1\langle 2\rangle$ and $a_2\langle 2\rangle$, but not, e.g., $a_1\langle 1\rangle$ and $a_2\langle 2\rangle$.

This invariant must also be heeded when we merge back together parts of the hypergraph: for example, if $a_1 R a_2$, we cannot merge back together

$a_1\langle 1\rangle$ and $a_1\langle 2\rangle$ but leave $a_2\langle 1\rangle$ and $a_2\langle 2\rangle$ split at the same time. Hence, what we must consider is merging all elements which are (in-)directly related by R simultaneously together. More succinctly, the objects considered to be merged must be the equivalence classes of the reflexive-transitive closure R^* of R, which we also call *split classes*.

4.1 Splitting

When we split the vertices of a hypergraph, we must take care that the created hyperedges respect the supplied split relation R, as explained above.

This is achieved by the following function split_R, which splits every node and introduces new hyperedges respecting R. Given a hypergraph G and split relation $R \subseteq V \times V$, we let $\mathrm{split}_R(G) = G'$ with

$$V' = \{a\langle b\rangle \mid a \in V, b \in \{1,2\}\},$$

$$E' = \{e\langle b_0,\ldots,b_k\rangle \mid\ e \in E, \mu(e) = a_0\cdots a_k,\ b_0,\ldots,b_k \in \{1,2\},\ a_i R a_j \text{ implies } b_i = b_j\},$$

$$\mu'(e\langle b_0,\ldots,b_k\rangle) = a_0\langle b_0\rangle\cdots a_k\langle b_k\rangle, \text{ where } \mu(e) = a_0\cdots a_k,$$

$$g' = g\langle 1\rangle, \text{ and}$$

$$p'(e\langle b_0,\ldots,b_k\rangle) = \frac{p(e)}{2^c} \text{ where } c = |\{e\langle \boldsymbol{b'}\rangle \in E' \mid \boldsymbol{b'} = b_0\cdots b_k\}|.$$

Note that the probabilities of hyperedges are distributed uniformly to their split copies in G'. In an implementation, these should be slightly randomized to give starting values for the EM algorithm, as mentioned by Petrov et al. (2006).

The split hypergraph's number of hyperedges is exponential in their arity. We can try to mitigate the problem of this exponential blowup by binarizing the trees of the TAG which was initially extracted from a corpus, following the description of Lang (1994).

4.2 Merging

As explained above, we have to merge all elements of a split class back together simultaneously. This will be denoted with the following function. Let $G' = \mathrm{split}_R(G)$ be a hypergraph resulting from a split, $R \subseteq V \times V$ a split relation, and $C \in V/R^*$ one of its split classes. The hypergraph which results from merging C back together is denoted by $G'' = \mathrm{merge}_R^C(G')$ with

$$V'' = V' \setminus \{a\langle b\rangle \mid a \in C, b \in \{1,2\}\} \cup C,$$

$$E'' = \{e\langle b'_0,\ldots,b'_k\rangle \mid\ e\langle b_0,\ldots,b_k\rangle \in E',\ \mu(e) = a_0\cdots a_k,\ a_i \in C \text{ implies } b'_i = 0,\ a_i \notin C \text{ implies } b'_i = b_i\},$$

where we write $o(e\langle b'_0,\ldots,b'_k\rangle) = e\langle b_0,\ldots,b_k\rangle$ for the relation of $e\langle b'_0,\ldots,b'_k\rangle$ to $e\langle b_0,\ldots,b_k\rangle$,

$$\mu''(e\langle b_0,\ldots,b_k\rangle) = a_0\langle b_0\rangle\cdots a_k\langle b_k\rangle, \text{ where } \mu(e) = a_0\cdots a_k,$$

$$g'' = \begin{cases} g & \text{if } g \in C \\ g' & \text{otherwise, and} \end{cases}$$

$$p''(e'') = \begin{cases} \displaystyle\sum_{e' \in o^{-1}(e'')} p'(e')/2 & \text{if } \mathrm{hd}(e) \in C \\ \displaystyle\sum_{e \in o^{-1}(e'')} p'(e') & \text{otherwise} \end{cases}$$

Note that the probabilities of hyperedges which are merged back together are summed up. In the case that a hyperedge's head vertex is merged, we have to normalize the resulting probability.

4.3 Treebank Corpora and Likelihood

During the split-merge procedure, we want to train the split hypergraph representations on a supplied treebank. Following the presentation of Prescher (2005), we will abstract away from the concrete data structures which might be used to represent a collection of trees, and just define a treebank corpus as a function $K\colon U_\Sigma \to \mathbb{R}_{\geq 0}$, assigning to every tree over an alphabet Σ a certain *frequency*, such that the set of trees with non-zero frequency is finite.

Let us assume that $G_0 = \mathrm{hg}(\mathcal{G})$ is the hypergraph representation of a PTAG $\mathcal{G}$ and G_n is the result of n split-merge cycles on G_0. Then we can define the likelihood of a corpus $K\colon U_\Sigma \to \mathbb{R}_{\geq 0}$ on G_n as

$$L(K \mid G_n) = \prod_{t \in U_\Sigma} \Big(\sum_{\substack{d \in H_{G_n} \\ \mathrm{yd}(u_n(d)) = t}} P(d \mid G_n)\Big)^{K(t)}$$

$$= \prod_{t \in U_\Sigma} \Big(\sum_{\substack{d' \in H_{G_0} \\ \mathrm{yd}(d') = t}} \big(\sum_{\substack{d \in H_{G_n} \\ u_n(d) = d'}} P(d \mid G_n)\big)\Big)^{K(t)}. \quad (1)$$

Hereby, the function u_n removes n levels of annotation from an annotated derivation, for every $n \in \mathbb{N}$. It is defined as the homomorphic extension of the function $\tilde{u}_n$ on hyperedges to unranked trees, where $\tilde{u}$ is defined for every hyperedge e by $\tilde{u}_0(e) = e$, and for $n > 1$, $\tilde{u}_n(e\langle \boldsymbol{b}\rangle) = \tilde{u}_{n-1}(e)$.

Algorithm 1 The state-split algorithm

Require: phg G, $n \in \mathbb{N}$, corpus $K \colon U_\Sigma \to \mathbb{R}_{\geq 0}$

1: **function** STATESPLIT(G, n, K)
2: $G_0 \leftarrow G$
3: **compute** split relation R_0
4: **for all** $i \in [n]$ **do**
5: $G_i \leftarrow$ SPLITMERGE(G_{i-1}, R_{i-1}, K)
6: **update** split relation R_{i-1} to R_{i-1}
7: **end for**
8: **return** G_n
9: **end function**

Note that in (1), for every $t \in U_\Sigma$ with non-zero frequency, there are only finitely many derivations $d' \in H_{G_0}$, which can be determined by employing a TAG parser at the beginning of the state-split process. For each of these d' we can compute the value of the innermost sum in a bottom-up manner, similarly to the computation of inside probabilities as described by Petrov et al. (2006).

In the following, given a hypergraph G_n, $n > 0$, we will identify $\mathrm{yd}(d)$ with $\mathrm{yd}(u_n(d))$, for every $d \in H_{G_n}$.

4.4 Overview of the Algorithm

Now we can denote the state-split algorithm for hypergraphs in pseudocode, cf. Alg. 1. After computing the initial split relation, the algorithm's outer loop (ll. 4–7) executes n split-merge cycles on G using the treebank corpus K for training. Additionally, in each step it updates the split relation to the newly generated hypergraph.

Note that the concrete computation of split classes depends strongly on the grammar formalism represented by the hypergraph, hence we leave it abstract in the general formulation of the algorithm. In our case, where we use tree-adjoining grammars as underlying formalism, we can instantiate it as follows: R_0 is defined as the finest symmetric relation R such that, for every vertex of the form $\mathsf{S}(t, y_i)$ in G_0, we have $(\mathsf{S}(t, y_i), \mathrm{lb}_t(y_i)^\star) \in R$. Similarly, given a split relation R_{i-1}, the updated split relation R_i is the finest symmetric relation R such that, for every $b \in \{0, 1, 2\}$ and pair $a_1\langle b\rangle$, $a_2\langle b\rangle$, if $a_1\langle b\rangle$ and $a_2\langle b\rangle$ are vertices in G_i such that $a_1 \, R_{i-1} \, a_2$, then $a_1\langle b\rangle \, R \, a_2\langle b\rangle$.

The split-merge cycle (cf. Alg. 2) can be considered as the core of the state-split algorithm. Supplied with a phg G, split relation R on G, and treebank corpus K, SPLITMERGE first of all splits the nodes of G, as defined in section 4.1. Then it

Algorithm 2 A split-merge cycle

Require: phg G, $R \subseteq V \times V$, corpus $K \colon U_\Sigma \to \mathbb{R}_{\geq 0}$

1: **function** SPLITMERGE(G, R, K)
2: $G' \leftarrow \mathrm{split}_R(G)$
3: $G' \leftarrow$ EMTRAIN(G', K)
4: **for all** $C \in V/R^*$ **do**
5: $G'' \leftarrow \mathrm{merge}_R^C(G')$
6: **if** $L(G'' \mid K)/L(G' \mid K) \geq \lambda$ **then**
7: $G' \leftarrow G''$
8: **end if**
9: **end for**
10: **return** G'
11: **end function**

uses the EM algorithm for maximum likelihood estimation on incomplete data to train the newly-split phg on the treebank K. Finally, in ll. 4–9, each split class is tentatively merged back together, and if the concomitant loss in likelihood does not fall below a certain factor $\lambda \in [0, 1]$, this merge is taken over permanently. As noted by Petrov et al., this combats the exponential blow-up of the hypergraph, as well as the phenomenon of overfitting.

For the sake of completeness, let us give a rough sketch of the EM algorithm for training state-split hypergraphs on treebank corpora, following the exposition of Prescher (2005). Given a phg G and a treebank corpus K, the EM algorithm alternatingly repeats two computations, called the *Expectation step (E-step)* and the *Maximization step (M-step)*, until the increase in likelihood of the newly-computed hypergraph on the corpus K falls beneath a certain threshold $\delta \in \mathbb{R}$.

Note that the EM algorithm is not guaranteed to find the actual global maximum of the likelihood, however, as already shown by Dempster et al. (1977), the likelihoods of the respective grammars are monotonically non-decreasing, and so at least a local maximum can be approximated.

In the algorithm's E-step, a complete-data corpus $C \colon H_G \to \mathbb{R}_{\geq 0}$ on the hyperpaths of G is generated by distributing the frequencies of every derived tree $t \in U_\Sigma$ in the corpus K to the hyperpaths representing derivations of t, weighted by their conditional probability given t. The M-step then uses this complete-data corpus to compute hyperedge probabilities by relative frequency estimation. Thereby, $\mathrm{ct}_e(d)$ denotes the number of appearances of the hyperedge e in the derivation d,

Algorithm 3 EM for (state-split) hypergraphs

Require: phg G, corpus $K\colon U_\Sigma \to \mathbb{R}_{\geq 0}$

function EMTRAIN(G, K)
  repeat
    $G' \leftarrow G$
    E-Step: define $C\colon H_G \to \mathbb{R}$ by:
    $C(d) = K(\mathrm{yd}(d)) \cdot P\big(d \mid \mathrm{yd}(d), G\big)$
    M-step: set new probabilities:
    for all $e \in E$ **do**
      $p(e) \leftarrow \frac{\sum_{d \in H_G} C(d) \cdot \mathrm{ct}_e(d)}{\sum_{d \in H_G} C(d) \cdot \mathrm{ct}_{\mathrm{hd}(e)}(d)}$
    end for
  until $L(G|K) - L(G'|K) < \delta$
  return G
end function

and $\mathrm{ct}_{\mathrm{hd}(e)}(d)$ the number of appearing hyperedges with the same head vertex as e.

One might remark that this concise, but formulaic presentation of EM is not immediately suitable for implementation, but, using the derivation employed by Gupta and Chen (2011), it is straightforward to bring it into the form of the well-known *Inside-Outside algorithm*, which has been adapted to PTAG by Schabes (1992).

5 State-Split preserves Hypergraph Representations

After all these definitions, we might ask ourselves the following question: given a hypergraph representation G of some PTAG $\mathcal{G}$, does the hypergraph which results from an application of the split-merge cycle to G still represent a PTAG? This question indeed arises quite naturally, after all it is an important requirement for the correctness of the state-split procedure for PTAGs with hypergraphs.

If we denote the result of a split-merge cycle on a phg G by $\mathrm{sm}(G)$, and the class of all hypergraph representations of probabilistic tree-adjoining grammars by $\mathcal{H}_\mathcal{T}$, then this question can essentially be answered by proving the inclusion

$$\mathrm{sm}(\mathcal{H}_\mathcal{T}) \subseteq \mathcal{H}_\mathcal{T}. \tag{2}$$

But the validity of this inclusion does certainly depend on the formal definition of $\mathcal{H}_\mathcal{T}$. Indeed, for the definition which comes to mind first, in which we just fix $\mathcal{H}_\mathcal{T}$ to be the set $\{\mathrm{hg}(\mathcal{G}) \mid \mathcal{G} \in \mathcal{T}\}$, i.e. the image of the class of all PTAGs under hg, the inclusion is *not* valid! This is due to the annotation of vertices and hyperedges in a split-merge cycle: hypergraphs that contain, for example, a vertex

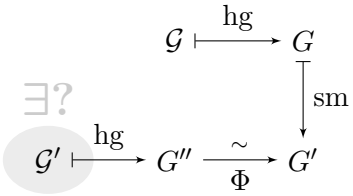

Figure 4: Proof idea

$A\langle 2 \rangle$, which was generated by splitting a vertex A, are arguably not in the image of hg!

However, one can show that the *structure* of a split-and-merged hypergraph still corresponds to the image of a PTAG $\mathcal{G}$ under hg. We will capture this structural identity by means of hypergraph isomorphisms.

A *hypergraph isomorphism* $\Phi\colon G \xrightarrow{\sim} G'$ is a tuple (Φ_1, Φ_2), where $\Phi_1\colon V \to V'$ and $\Phi_2\colon E \to E'$ are bijections such that $\Phi_1(g) = g'$ and $\mu'(\Phi_2(e)) = \Phi_1(a_0) \cdots \Phi_1(a_n)$ if $\mu(e) = a_0 \cdots a_n$. We write $G \cong G'$ if there is an isomorphism $\Phi\colon G \xrightarrow{\sim} G'$.

We define the set of all possible hypergraph representations of PTAG as

$$\mathcal{H}_\mathcal{T} = \{G \in \mathcal{H} \mid \exists \mathcal{G} \in \mathcal{T}.\, \mathrm{hg}(\mathcal{G}) \cong G\}.$$

Obviously, for every PTAG $\mathcal{G}$, $\mathrm{hg}(\mathcal{G}) \in \mathcal{H}_\mathcal{T}$, i.e., it is indeed a hypergraph representation according to this definition. As visualized in Fig. 4, the core of the proof of (2) is then as follows: Given $\mathcal{G} \in \mathcal{T}$, $G \in \mathcal{H}$, and $G' = \mathrm{sm}(G)$, we have to construct a PTAG $\mathcal{G}'$ and hypergraph isomorphism $\Phi\colon \mathrm{hg}(\mathcal{G}') \xrightarrow{\sim} G'$.

We construct $\mathcal{G}'$ so that the trees of $\mathcal{G}'$ are relabelings of those in $\mathcal{G}$, generated by incorporating the annotations to the hyperedges in G to substitution and adjoining sites:

$$\mathcal{S}' = \{t\langle \boldsymbol{b} \rangle \mid \mathsf{s}(t)\langle \boldsymbol{b} \rangle \in E'\},$$
$$\mathcal{A}' = \{t\langle \boldsymbol{b} \rangle \mid \mathsf{a}(t)\langle \boldsymbol{b} \rangle \in E'\},$$

where $t\langle a\, b_1 \cdots b_{\mathrm{rk}_1(t)}\, c_1 \cdots c_{\mathrm{rk}_2(t)} \rangle$ is the result of replacing the root symbol A of t by $A\langle a \rangle$, every substitution site A_{x_i} in t by $A\langle b_i \rangle_{x_i}$, and every adjoining site A_{y_j} by $A\langle c_j \rangle_{y_j}$, for $i \in [\mathrm{rk}_1(t)]$ and $j \in [\mathrm{rk}_2(t)]$. The probabilities $P(t)$ of elementary trees t, as well as the activation probabilities $Q_t(y_i)$ are just read off from $p(\mathsf{s}(t))$, $p(\mathsf{a}(t))$, resp. $p(\mathsf{y}(t, y_i))$.

The hypergraph isomorphism Φ then just reverses this relabeling. Given nodes or hyperedges from $\mathrm{hg}(\mathcal{G}')$ with annotations in them, it removes them from the contained symbols resp. elementary

trees and moves them "to the back", i.e.,

$$\Phi_1(A\langle b\rangle) = A\langle b\rangle, \quad \Phi_1\big((A\langle b\rangle)^\star\big) = A^\star\langle b\rangle,$$
$$\Phi_1\big(\mathsf{S}(t\langle \boldsymbol{b}\rangle, y_i)\big) = \mathsf{S}(t, y_i)\langle b^{(i)}\rangle$$

and

$$\Phi_2\big(\mathsf{s}(t\langle \boldsymbol{b}\rangle)\big) = \mathsf{s}(t)\langle \boldsymbol{b}\rangle,$$
$$\Phi_2\big(\mathsf{s}(t\langle \boldsymbol{b}\rangle)\big) = \mathsf{s}(t)\langle \boldsymbol{b}\rangle,$$
$$\Phi_2\big(\mathsf{y}(t\langle \boldsymbol{b}\rangle, y_i)\big) = \mathsf{y}(t, y_i)\langle b^{(i)} b^{(i)}\rangle,$$
$$\Phi_2\big(\mathsf{n}(t\langle \boldsymbol{b}\rangle, y_i)\big) = \mathsf{n}(t, y_i)\langle b^{(i)}\rangle,$$

where, in all three cases, $b^{(i)} = \boldsymbol{b}(\mathrm{rk}_1(t) + i)$.

Now, Φ can indeed be proven to be a hypergraph isomorphism between $\mathcal{G}'$ and $\mathrm{sm}(G)$, but for reasons of space, we will omit these details from this work. Note that the construction of $\mathcal{G}'$ and Φ can alternatively be interpreted as the definition of a *read-off* procedure, which allows our system *Vanda* to convert back its internal state-split hypergraph representation into a probabilistic tree-adjoining grammar, e.g. to display the resulting grammars for means of debugging. Thus, it also has a hands-on relevance for implementation.

Acknowledgements We thank the anonymous reviewers for their valuable comments, and our colleagues Toni Dietze and Matthias Büchse for fruitful discussions.

References

Yehoshua Bar-Hillel, Micha A Perles, and Eli Shamir. 1961. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, 14:143–172.

Matthias Büchse, Mark-Jan Nederhof, and Heiko Vogler. 2011. Tree Parsing with Synchronous Tree-Adjoining Grammars. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 14–25, Dublin, Ireland, October. Association for Computational Linguistics.

Matthias Büchse, Toni Dietze, Johannes Osterholzer, Anja Fischer, and Linda Leuschner. 2012. Vanda – A Statistical Machine Translation Toolkit. In *Proceedings of the 6th International Workshop Weighted Automata: Theory and Applications*, pages 36–37.

John Chen, Srinivas Bangalore, and K Vijay-Shanker. 2006. Automated Extraction of Tree-Adjoining Grammars from Treebanks. *Natural Language Engineering*, 12(3):251.

Arthur P Dempster, Nan McKenzie Laird, and Donald B Rubin. 1977. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society*, B(39):1–38.

Maya R Gupta and Yihua Chen. 2011. Theory and Use of the EM Algorithm. *Foundations and Trends in Signal Processing*, 4(3):223–296.

Aravind K Joshi and Yves Schabes. 1991. Tree-Adjoining Grammars and Lexicalized Grammars. In Maurice Nivat and Andreas Podelski, editors, *Definability and Recognizability of Sets of Trees*, pages 409–431. Elsevier.

Dan Klein and Christopher D Manning. 2004. Parsing and Hypergraphs. In Harry Bunt, John Carroll, and Giorgio Satta, editors, *New Developments in Parsing Technology*, volume 23 of *Text, Speech and Language Technology*, pages 351–372. Springer Netherlands.

Bernard Lang. 1994. Recognition can be Harder than Parsing. *Computational Intelligence*, 10(4):486–494.

Andreas Maletti and Giorgio Satta. 2009. Parsing Algorithms based on Tree Automata. In Harry Bunt, editor, *Proceedings of the 11th International Conference on Parsing Technologies*, pages 1–12. Association for Computational Linguistics.

Andreas Maletti. 2010. A Tree Transducer Model for Synchronous Tree-Adjoining Grammars. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1067–1076. Association for Computational Linguistics.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning Accurate, Compact, and Interpretable Tree Annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440, Sydney, Australia. Association for Computational Linguistics.

Detlef Prescher. 2005. A Tutorial on the Expectation-Maximization Algorithm Including Maximum-Likelihood Estimation and EM Training of Probabilistic Context-Free Grammars. Technical report, 15th European Summer School in Logic, Language, and Information.

Philip Resnik. 1992. Probabilistic Tree-Adjoining Grammar As A Framework For Statistical Natural Language Processing. In *Proceedings of the 14th Conference on Computational Linguistics*, pages 418–424.

James Rogers. 2003. wMSO theories as grammar formalisms. *Theoretical Computer Science*, 293(2):291–320, February.

Yves Schabes. 1992. Stochastic Tree-Adjoining Grammars. In *Proceedings of the Workshop on Speech and Natural Language*, HLT '91, pages 140–145, Stroudsburg, PA, USA. Association for Computational Linguistics.

Hiroyuki Shindo, Yusuke Miyao, Akinori Fujino, and Masaaki Nagata. 2012. Bayesian Symbol-Refined Tree Substitution Grammars for Syntactic Parsing. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, pages 440–448.