Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence

Statistical Parsing with Probabilistic Symbol-Refined Tree Substitution Grammars*

Hiroyuki Shindo† **Yusuke Miyao**‡ **Akinori Fujino**† **Masaaki Nagata**†
†NTT Communication Science Laboratories, NTT Corporation
2-4 Hikaridai, Seika-cho, Soraku-gun, Kyoto, Japan
{*shindo.hiroyuki,fujino.akinori,nagata.masaaki*}*@lab.ntt.co.jp*
‡National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan
yusuke@nii.ac.jp

Abstract

We present probabilistic Symbol-Refined Tree Substitution Grammars (SR-TSG) for statistical parsing of natural language sentences. An SR-TSG is an extension of the conventional TSG model where each nonterminal symbol can be refined (subcategorized) to fit the training data. Our probabilistic model is consistent based on the hierarchical Pitman-Yor Process to encode backoff smoothing from a fine-grained SR-TSG to simpler CFG rules, thus all grammar rules can be learned from training data in a fully automatic fashion. Our SR-TSG parser achieves the state-of-the-art performance on the Wall Street Journal (WSJ) English Penn Treebank data.

1 Introduction

Parsing is a fundamental task in natural language processing (NLP). Many NLP parsers take a statistical approach to resolve structural ambiguity of sentences. Specifically, they assume probabilistic grammars for modeling syntax trees and make use of treebank data (i.e., a corpus of hand-annotated parse trees) for parameter estimation. It has been shown that probabilistic context-free grammars (CFG) extracted from treebank data perform poorly when nonterminal symbols are coarse due to unrealistic context freedom assumptions [Klein and Manning, 2003].

There has been an increasing interest in tree substitution grammars (TSG) as an alternative to CFG [Post and Gildea, 2009; Tenenbaum *et al.*, 2009; Cohn *et al.*, 2010]. TSG is a natural extension of CFG in which nonterminal symbols can be rewritten with arbitrarily large tree fragments. These tree fragments have great advantages over tiny CFG rules since they can capture non-local contexts explicitly such as predicate-argument structures, idioms and grammatical agreements [Cohn *et al.*, 2010]. One major drawback of TSG is that the context freedom assumptions still remain, that is, tree fragments are generated that are conditionally independent of all others given coarse root nonterminal symbols.

To address the TSG problem, we propose Symbol-Refined Tree Substitution Grammars (SR-TSG). SR-TSG is an extension of the conventional TSG model where each nonterminal symbol can be refined (subcategorized) to fit the training data. Symbol refinement is a commonly used technique for high-performance CFG parsers, however, our work differs from previous studies in that we focus on a unified model where TSG rules and symbol subcategories are learned from training data in a fully automatic and consistent fashion. In order to achieve our purpose, we design the probabilistic SR-TSG model based on the hierarchical Pitman-Yor Process (PYP) [Pitman and Yor, 1997] to encode backoff smoothing from a fine-grained SR-TSG to simpler CFG rules. Our SR-TSG parser successfully achieves an F1 score of 92.4% in the WSJ English Penn Treebank parsing task, which is a 7.7 point improvement over a conventional TSG parser.

2 Symbol-Refined Tree Substitution Grammars

2.1 Background

A derivation of SR-TSG is a process of forming a parse tree. It starts with a root symbol and rewrites nonterminal symbols with elementary trees (i.e., tree fragments of height ≥ 1) until all leaf nodes become lexical words. Figure **??** shows an example parse tree and Figure **??** shows its example SR-TSG derivation. Since different derivations may produce the same parse tree, we need a probabilistic model of SR-TSG and predicts derivations from observed parse trees in an unsupervised way.

A probabilistic SR-TSG assigns a probability to each elementary tree. The probability of a derivation is simply defined as the product of the probabilities of its component elementary trees. The posterior distribution over elementary trees $\mathbf{e}$ given a parse tree t can be computed as follows: $p(\mathbf{e}|t) \propto p(t|\mathbf{e})\,p(\mathbf{e})$ where $p(t|\mathbf{e})$ is either equal to 1 (when t and $\mathbf{e}$ are consistent) or 0 (otherwise). Therefore,

*The paper on which this extended abstract is based was the recipient of the best paper award of the 2013 Association for Computational Linguistics [Shindo *et al.*, 2012].

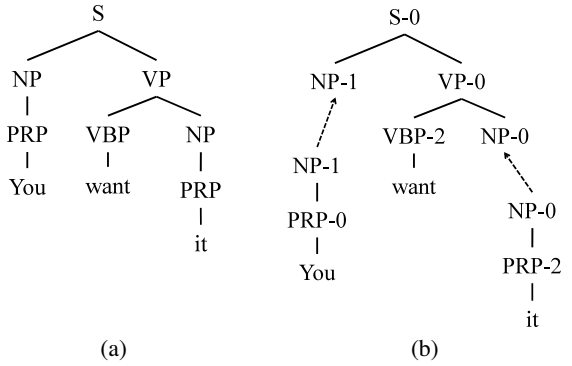

Figure 1: (a) Example parse tree. (b) Example SR-TSG derivation of (a). The refinement annotation is hyphenated with a nonterminal symbol.

the task of SR-TSG induction from parse trees turns out to consist of modeling the prior distribution $p(\mathbf{e})$.

2.2 Probabilistic Model

We propose a probabilistic model of SR-TSG, $p(\mathbf{e})$, based on the Pitman-Yor Process (PYP) [Pitman and Yor, 1997]. One major issue as regards modeling an SR-TSG is that the space of the grammar rules will be very sparse since SR-TSG allows for arbitrarily large tree fragments and also an arbitrarily large set of symbol subcategories. We address this data sparseness problem by employing three-level hierarchical PYP to encode a backoff scheme from a set of complex SR-TSG rules to that of simpler CFG rules. Table 2 shows an example of the SR-TSG rule and its backoff tree fragments as an illustration of this three-level hierarchy.

The first level is a distribution over the SR-TSG rules as follows.

$$\begin{aligned} e\,|x_k &\sim G_{x_k} \\ G_{x_k} &\sim \text{PYP}\left(d_{x_k}, \theta_{x_k}, P^{\text{sr-tsg}}(\cdot\,|x_k)\right), \end{aligned}$$

where e is an elementary tree rooted with symbol x_k. We represent a raw nonterminal symbol in the corpus as x and refined (subcategorized) symbol as x_k where $k = 0, 1, \ldots$ is an index of the symbol subcategory. Suppose x is NP and its symbol subcategory is 0, then x_k is NP_0. $P^{\text{sr-tsg}}(\cdot\,|x_k)$ is a *base distribution* over infinite space of symbol-refined elementary trees rooted with x_k, which provides the backoff probability of e. The remaining parameters d_{x_k} and θ_{x_k} control the strength of the base distribution.

The backoff probability $P^{\text{sr-tsg}}(e\,|x_k)$ is computed by the product of *symbol-refined CFG (SR-CFG)* rules that e contains as follows.

$$\begin{aligned} P^{\text{sr-tsg}}(e\,|x_k) &= \prod_{f \in F(e)} s_{c_f} \times \prod_{i \in I(e)} (1 - s_{c_i}) \\ &\times H\left(\text{cfg-rules}(e\,|x_k)\right) \\ \alpha\,|x_k &\sim H_{x_k} \\ H_{x_k} &\sim \text{PYP}\left(d_x, \theta_x, P^{\text{sr-cfg}}(\cdot\,|x_k)\right), \end{aligned}$$

where $F(e)$ is a set of frontier nonterminal nodes and $I(e)$ is a set of internal nodes in e. c_f and c_i are nonterminal symbols of nodes f and i, respectively. s_c is the probability of stopping the expansion of a node labeled with c. SR-CFG rules are CFG rules where every symbol is refined, as shown in Table 2 (b). The function cfg-rules$(e\,|x_k)$ returns the SR-CFG rules that e contains, which take the form of $x_k \to \alpha$. Each SR-CFG rule α rooted with x_k is drawn from the backoff distribution H_{x_k}, and H_{x_k} is produced by PYP. This distribution over the SR-CFG rules forms the second level hierarchy of our model.

The third level of our model, $P^{\text{sr-cfg}}(\alpha\,|x_k)$, is a distribution over the *root-unrefined CFG (RU-CFG)* rule as follows.

$$\begin{aligned} P^{\text{sr-cfg}}(\alpha\,|x_k) &= I\left(\text{root-unrefine}(\alpha\,|x_k)\right) \\ \alpha\,|x &\sim I_x \\ I_x &\sim \text{PYP}\left(d'_x, \theta'_x, P^{\text{ru-cfg}}(\cdot\,|x)\right), \end{aligned}$$

where the function root-unrefine$(\alpha\,|x_k)$ returns the RU-CFG rule of α, which takes the form of $x \to \alpha$. The RU-CFG rule is a CFG rule where the root symbol is unrefined and all leaf nonterminal symbols are refined, as shown in Table 2 (c). As well as the second level hierarchy, each RU-CFG rule α rooted with x is drawn from the backoff distribution I_x, and I_x is produced by PYP. Finally, we set the backoff probability of the RU-CFG rule, $P^{\text{ru-cfg}}(\alpha\,|x)$, so that it is uniform as follows.

$$P^{\text{ru-cfg}}(\alpha\,|x) = \frac{1}{|x \to \cdot|}.$$

where $|x \to \cdot|$ is the number of RU-CFG rules rooted with x. Overall, our hierarchical model encodes backoff smoothing consistently from the SR-TSG rules to the SR-CFG rules, and from the SR-CFG rules to the RU-CFG rules. Therefore, SR-TSG rules not seen in the training corpus will never be given a probability of zero.

2.3 Inference

For the inference of SR-TSG rules from treebank data, We use the Markov Chain Monte Carlo (MCMC) method to obtain derivation samples from the posterior distribution $p(\mathbf{e}\,|\mathbf{t}, \mathbf{d}, \boldsymbol{\theta}, \mathbf{s})$. The inference of the SR-TSG derivations corresponds to inducing two kinds of latent variables: symbol subcategories and substitution sites. Substitution site is a node of parse tree which forms the root node of some elementary tree. For example, S_0, NP_1 and NP_0 in Figure **??** are substitution sites, and all other nodes are non-substitution sites. We first infer latent symbol subcategories of every nodes, then infer latent substitution sites stepwise. During the inference of symbol subcategories, every internal node is fixed as a substitution site, i.e., we assume SR-CFG rather than SR-TSG. After that, we unfix that assumption and infer latent substitution sites given symbol-refined parse trees. This stepwise learning is simple and efficient in practice, but we believe that the joint learning of both latent variables is possible. We will deal with this in future work. Here we describe each inference algorithm in detail.

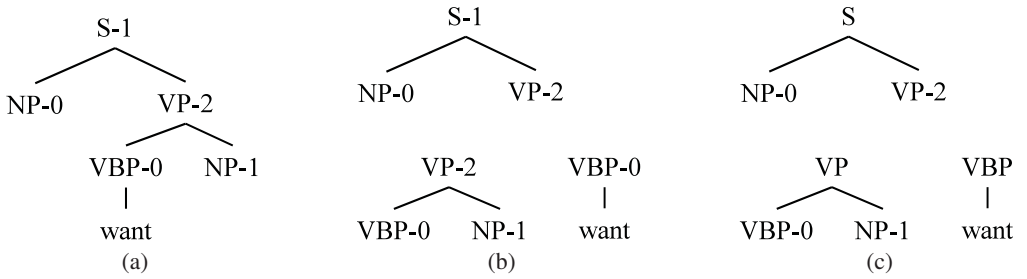

Figure 2: Example three-level backoff.. (a) Example SR-TSG rule. (b) SR-CFG rules of (a). (c) RU-CFG rules of (b).

Inference of Symbol Subcategories

We adopt split and merge training [Petrov *et al.*, 2006] for the inference of latent symbol subcategories. In each split-merge step, each symbol is split into at most two subcategories. For example, every NP symbol in the training data is split into either NP_0 or NP_1 to maximize the posterior probability. After convergence, we measure the loss of each split symbol in terms of the likelihood incurred when removing it, then the smallest 50% of the newly split symbols as regards that loss are merged to avoid overfitting. The split-merge algorithm terminates when the total number of steps reaches the user-specified value.

In each splitting step, we use two types of blocked MCMC algorithm: the *sentence-level* blocked Metroporils-Hastings (MH) sampler and the *tree-level* blocked Gibbs sampler, while [Petrov *et al.*, 2006] use a different MLE-based model and the EM algorithm. Our sampler iterates sentence-level sampling and tree-level sampling alternately.

The sentence-level MH sampler is a recently proposed algorithm for grammar induction [Johnson *et al.*, 2007b; Cohn *et al.*, 2010]. In this work, we apply it to the training of symbol splitting. The MH sampler consists of the following three steps: for each sentence, 1) calculate the inside probability [Lari and Young, 1991] in a bottom-up manner, 2) sample a derivation tree in a top-down manner, and 3) accept or reject the derivation sample by using the MH test. See [Cohn *et al.*, 2010] for details. This sampler simultaneously updates blocks of latent variables associated with a sentence, thus it can find MAP solutions efficiently.

The tree-level blocked Gibbs sampler focuses on the type of SR-TSG rules and simultaneously updates all root and child nodes that are annotated with the same SR-TSG rule. For example, the sampler collects all nodes that are annotated with $S_0 \rightarrow NP_1VP_2$, then updates those nodes to another subcategory such as $S_0 \rightarrow NP_2VP_0$ according to the posterior distribution. This sampler is similar to table label resampling [Johnson and Goldwater, 2009], but differs in that our sampler can update multiple table labels simultaneously when multiple tables are labeled with the same elementary tree. The tree-level sampler also simultaneously updates blocks of latent variables associated with the type of SR-TSG rules, thus it can find MAP solutions efficiently.

Inference of Substitution sites

After the inference of symbol subcategories, we use the standard Gibbs sampling to infer the substitution sites of parse trees as described in [Cohn and Lapata, 2009; Post and Gildea, 2009]. We assign a binary variable to each internal node in the training data, which indicates whether that node is a substitution site (1) or not (0). For each iteration, the Gibbs sampler works by sampling the value of each binary variable in random order according to the posterior distribution.

During the inference, our sampler ignores the symbol subcategories of internal nodes of elementary trees since they do not affect the TSG derivations. For example, the elementary trees "$(S_0\ (NP_0\ NNP_0)\ VP_0)$" and "$(S_0\ (NP_1\ NNP_0)\ VP_0)$" are regarded as being the same when we calculate the probability of tree fragments. In practice, this heuristics is helpful for fixing symbol over-splitting of previous training step and also finding large tree fragments.

3 Experiment

3.1 Setting

We conducted parsing experiments on the Wall Street Journal (WSJ) portion of the English Penn Treebank data set [Marcus *et al.*, 1993] with standard data splitting: sections 2-21 for training, 22 for development and 23 for testing. We also used section 2 as a *small* training set for evaluating the performance of our model under low-resource conditions. Henceforth, we distinguish the small training set (section 2) from the full training set (sections 2-21). All parse trees in the data set are right-binarized to make grammar rules with only unary and binary productions. Furthermore, we replaced lexical words with count ≤ 5 in the training data with one of 50 unknown words using lexical features, following [Petrov *et al.*, 2006]. We also split off all the function tags and eliminated empty nodes from the data set, following [Johnson, 1998]. The parsing results were obtained with the MAX-RULE-PRODUCT algorithm [Petrov *et al.*, 2006] by using the SR-TSG rules extracted from our model. Following the previous work, we evaluated the accuracy of our parser by bracketing F1 score of predicted parse trees.

3.2 Results and Discussion

Comparison of SR-TSG with TSG

We compared the parsing accuracy of SR-TSG model with that of CFG and TSG models. We also tested our model with three backoff hierarchy settings to evaluate the effects of backoff smoothing on parsing accuracy. Table 1 shows the F1 scores of the CFG, TSG and SR-TSG parsers for small and full training sets. In Table 1, SR-TSG ($P^{\text{sr-tsg}}$) denotes

Model	F1 (small)	F1 (full)
CFG	61.9	63.6
TSG	77.1	85.0
SR-TSG ($P^{\text{sr-tsg}}$)	73.0	86.4
SR-TSG ($P^{\text{sr-tsg}}$, $P^{\text{sr-cfg}}$)	79.4	89.7
SR-TSG ($P^{\text{sr-tsg}}$, $P^{\text{sr-cfg}}$, $P^{\text{ru-cfg}}$)	**81.7**	**91.1**

Table 1: Comparison of parsing accuracy with the small and full training sets.

Model	F1
TSG (no symbol refinement)	
Post and Gildea (2009)	*82.6
Cohn et al. (2010)	84.7
TSG with Symbol Refinement	
Bansal et al. (2010)	88.1
SR-TSG (single)	**91.1**
SR-TSG (multiple)	**92.4**
CFG with Symbol Refinement	
Petrov and Klein (2007)	90.1
Petrov (2010)	91.8
Discriminative	
Charniak and Johnson (2005)	91.4
Huang (2008)	91.7

Table 2: Our parsing performance for the testing set compared with those of other parsers. *Results for the sentence length ≤ 40.

that we used only the topmost level of the hierarchy. Similary, SR-TSG ($P^{\text{sr-tsg}}$, $P^{\text{sr-cfg}}$) denotes that we used only the $P^{\text{sr-tsg}}$ and $P^{\text{sr-cfg}}$ backoff models.

Our best model, SR-TSG ($P^{\text{sr-tsg}}$, $P^{\text{sr-cfg}}$, $P^{\text{ru-cfg}}$), outperformed both the CFG and TSG models on both training sets. It suggests that the conventional TSG model trained from the vanilla treebank is insufficient to resolve structural ambiguities caused by coarse symbol annotations in a training corpus. As we expected, symbol refinement can be helpful with the TSG model for further fitting the training set and improving the parsing accuracy.

The performance of the SR-TSG parser was strongly affected by its backoff models. For example, the simplest model, $P^{\text{sr-tsg}}$, performed poorly compared with our best model. This result suggests that the SR-TSG rules extracted from the training set are very sparse and cannot cover the space of unknown syntax patterns in the testing set. Therefore, well-designed backoff modeling is essential for the SR-TSG parser.

Comparison of SR-TSG with Other Models

We compared the accuracy of the SR-TSG parser with that of conventional high-performance parsers. Table 2 shows the F1 scores of an SR-TSG and conventional parsers with the full training set.

Our SR-TSG (single) parser achieved an F1 score of 91.1%, which is a 6.4 point improvement over the conventional Bayesian TSG parser reported by [Cohn *et al.*, 2010]. Our model can be viewed as an extension of Cohn's work by the incorporation of symbol refinement. Therefore, this result confirms that a TSG and symbol refinement work complementarily in improving parsing accuracy. The work of [Post and Gildea, 2009] is also based on Bayesian modeling of TSG, however, they do not binarize the treebank data.

Compared with a symbol-refined CFG model such as the Berkeley parser [Petrov *et al.*, 2006], the SR-TSG model can use large tree fragments, which strengthens the probability of frequent syntax patterns in the training set. Indeed, the few very large rules of our model memorized full parse trees of sentences, which were repeated in the training set.

In Table 2, SR-TSG (multiple) is a combination of multiple SR-TSG models, following the work of [Petrov, 2010]. The parsing results are obtained as follows. We first trained sixteen SR-TSG models independently and produced a 100-best list of the derivations for each model. Then, we erased the subcategory information of parse trees and selected the best tree that achieved the highest likelihood under the product of sixteen models. The SR-TSG (multiple) achieved an F1 score of 92.4%, which is a state-of-the-art result for the WSJ parsing task. Compared with discriminative reranking parsers ([Charniak and Johnson, 2005] and [Huang, 2008]), combining multiple grammars by using the product model provides the advantage that it does not require any additional training. Several studies [Fossum and Knight, 2009; Zhang *et al.*, 2009] have proposed different approaches that involve combining k-best lists of candidate trees. We will deal with those methods in future work.

4 Related Work

Several studies have combined TSG with symbol refinement. An adaptor grammar [Johnson *et al.*, 2007a] is a sort of nonparametric Bayesian TSG model with symbol refinement, and is thus closely related to our SR-TSG model. However, an adaptor grammar differs from ours in that all its rules are *complete*: all leaf nodes must be terminal symbols, while our model permits nonterminal symbols as leaf nodes. Furthermore, adaptor grammars have largely been applied to the task of unsupervised structural induction from raw texts such as morphology analysis, word segmentation [Johnson and Goldwater, 2009] rather than constituent syntax parsing.

An all-fragments grammar [Bansal and Klein, 2010] is another variant of TSG that aims to utilize all possible subtrees as rules. It maps a TSG to an implicit representation to make the grammar tractable and practical for large-scale parsing. The manual symbol refinement described in [Klein and Manning, 2003] was applied to an all-fragments grammar and this improved accuracy in the English WSJ parsing task. As mentioned in the introduction, our model focuses on the automatic

learning of a TSG and symbol refinement without heuristics.

5 Conclusion

We have presented an SR-TSG, which is an extension of the conventional TSG model where each symbol of tree fragments can be automatically subcategorized to fit the training data. Our probabilistic model is based on the three-level PYP to handle with data sparseness problem. We confirmed our model significantly outperformed the conventional TSG and achieved state-of-the-art result in a WSJ parsing task.

References

[Bansal and Klein, 2010] Mohit Bansal and Dan Klein. Simple, Accurate Parsing with an All-Fragments Grammar. In *In Proc. of ACL*, pages 1098–1107, 2010.

[Charniak and Johnson, 2005] Eugene Charniak and Mark Johnson. Coarse-to-Fine n-Best Parsing and MaxEnt Discriminative Reranking. *In Proc. of ACL*, 1:173–180, 2005.

[Cohn and Lapata, 2009] Trevor Cohn and Mirella Lapata. Sentence Compression as Tree Transduction. *Journal of Artificial Intelligence Research*, 34:637–674, 2009.

[Cohn *et al.*, 2010] Trevor Cohn, Phil Blunsom, and Sharon Goldwater. Inducing Tree-Substitution Grammars. *Journal of Machine Learning Research*, 11:3053–3096, 2010.

[Fossum and Knight, 2009] Victoria Fossum and Kevin Knight. Combining Constituent Parsers. *In Proc. of HLT-NAACL*, pages 253–256, 2009.

[Huang, 2008] Liang Huang. Forest Reranking : Discriminative Parsing with Non-Local Features. *In Proc. of ACL*, 19104:0, 2008.

[Johnson and Goldwater, 2009] Mark Johnson and Sharon Goldwater. Improving nonparameteric Bayesian inference: experiments on unsupervised word segmentation with adaptor grammars. In *In Proc. of HLT-NAACL*, pages 317–325, 2009.

[Johnson *et al.*, 2007a] Mark Johnson, Thomas L Griffiths, and Sharon Goldwater. Adaptor Grammars : A Framework for Specifying Compositional Nonparametric Bayesian Models. *Advances in Neural Information Processing Systems 19*, 19:641–648, 2007.

[Johnson *et al.*, 2007b] Mark Johnson, Thomas L Griffiths, and Sharon Goldwater. Bayesian Inference for PCFGs via Markov chain Monte Carlo. In *In Proc. of HLT-NAACL*, pages 139–146, 2007.

[Johnson, 1998] Mark Johnson. PCFG Models of Linguistic Tree Representations. *Computational Linguistics*, 24:613–632, 1998.

[Klein and Manning, 2003] Dan Klein and Christopher D Manning. Accurate Unlexicalized Parsing. *In Proc. of ACL*, 1:423–430, 2003.

[Lari and Young, 1991] K Lari and S J Young. Applications of Stochastic Context-Free Grammars Using the Inside–Outside Algorithm. *Computer Speech and Language*, 5:237–257, 1991.

[Marcus *et al.*, 1993] Mitchell P Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19:313–330, 1993.

[Petrov *et al.*, 2006] Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning Accurate, Compact, and Interpretable Tree Annotation. *In Proc. of ACL*, pages 433–440, 2006.

[Petrov, 2010] Slav Petrov. Products of Random Latent Variable Grammars. *In Proc. of HLT-NAACL*, pages 19–27, 2010.

[Pitman and Yor, 1997] Jim Pitman and Marc Yor. The two-parameter Poisson-Dirichlet distribution derived from a stable subordinator. *The Annals of Probability*, 25:855–900, 1997.

[Post and Gildea, 2009] Matt Post and Daniel Gildea. Bayesian Learning of a Tree Substitution Grammar. In *In Proc. of ACL-IJCNLP*, pages 45–48, 2009.

[Shindo *et al.*, 2012] Hiroyuki Shindo, Yusuke Miyao, Akinori Fujino, Masaaki Nagata. Bayesian Symbol-Refined Tree Substitution Grammars for Syntactic Parsing. *In Proc. of ACL*, pages 440–448, 2012.

[Tenenbaum *et al.*, 2009] J Tenenbaum, TJ O'Donnell, and ND Goodman. Fragment Grammars: Exploring Computation and Reuse in Language. *MIT Computer Science and Artificial Intelligence Laboratory Technical Report Series*, 2009.

[Zhang *et al.*, 2009] Hui Zhang, Min Zhang, Chew Lim Tan, and Haizhou Li. K-Best Combination of Syntactic Parsers. *In Proc. of EMNLP*, pages 1552–1560, 2009.