A Generative Probability Model for Unification-Based Grammars

Helmut Schmid
Institute for Computational Linguistics
University of Stuttgart
Azenbergstraße 12, 70174 Stuttgart, Germany
schmid@ims.uni-stuttgart.de

Abstract

A generative probability model for unification-based grammars is presented in which rule probabilities depend on the feature structure of the expanded constituent. The presented model is the first model which requires no normalization and allows the application of dynamic programming algorithms for disambiguation (Viterbi) and training (Inside-Outside). Another advantage is the small number of parameters.

1 Introduction

A number of probability models for unification-based grammar (UG) formalisms have been proposed in the past. Some models use PCFG approximations (Eisele, 1994; Brew, 1995; Kiefer et al., 2002), others are based on Random Fields (Abney, 1997) or the closely related log-linear models (Johnson et al., 1999; Riezler et al., 2000; Stefan Riezler and Johnson, 2002). All these models require a normalization of the parse probabilities in order to obtain a distribution over all parses which sums to 1. However, the computation of the normalization constant seems only tractable for conditional models (Johnson et al., 1999) which define the probability of parses given their yields. Such models are useful for syntactic disambiguation, but not e.g. for language modeling.

This paper presents a generative probability model which models a stochastic generator for UGs. The probability of a parse tree is decomposed into a product of rule probabilities which depend on the previously instantiated features of the expanded constituents. Parse probabilities will sum to 1 unless dead end derivations exist. So, normalization is not required. The well-known algorithms for PCFGs (Viterbi, Inside-Outside) are applicable which facilitates efficient implementations of the model.

2 Background

2.1 PCFG Approximation Models

PCFG approximation approaches (Eisele, 1994; Brew, 1995; Kiefer et al., 2002) generalize UGs to context-free grammars which generate a superset of the parses of the UG. In the simplest case (Eisele, 1994), the UG rules are generalized by dropping the feature constraints. The CFG is turned into a PCFG and trained (e.g. on a treebank). The trained PCFG is used to assign probabilities to the analyses of the UG and to disambiguate between them.

The PCFG defines a probability distribution over the CFG parses. Since the UG generates fewer parses than the CFG, the probabilities of the UG parses will sum to less than 1. In order to get a probability distribution over UG parses, the parse probabilities are normalized by dividing by the sum of the probabilities of all UG parses. However, this sum is not computable for UGs generating an infinite number of parses.

The PCFG approximation approach faces another problem. Estimation of rule probabilities with relative frequency estimates is not a consistent training method for renormalized PCFG approximation models as the following example[1] shows. Consider the grammar[2] in figure 1 whose context-free backbone is trained on the treebank in figure 2. The relative frequency estimates of the probabilities for rules 1 through 5 are 1, 2/3, 1/3, 2/3 and 1/3, respectively. The probability assigned to the first parse tree in the treebank is $1 * 2/3 * 2/3 = 4/9$ and the probability assigned to the second parse tree is $1*1/3*1/3 = 1/9$. So,

[1] A similar example was presented in (Abney, 1997).

[2] This grammar is written in the YAP formalism (Schmid, 2000). On the right-hand side of feature equations, capitalized names are variables and lower-case names are values. The head of a rule is marked with a backquote.

the first parse tree is four times more likely than the second according to the probability model, although its frequency in the treebank is only twice as high. If parse probabilities are normal-

```
NP{} -> DT{Number=N;} 'N{Number=N;};  (1)
"this"  : DT {Number=sg;};            (2)
"these" : DT {Number=pl;};            (3)
"man"   : N {Number=sg;};             (4)
"men"   : N {Number=pl;};             (5)
```

Figure 1: a simple constraint grammar

ized, the estimated parameters are not even optimal: The sum of the probabilities of all valid parses in the above example is 5/9. The renormalized probabilities of the two valid parses are therefore 1/5 and 4/5 and the probability of the training corpus is 4/5*4/5*1/5=16/125=0.128. If we set e.g. the probability of rules 2 and 4 to 0.6 and the probability of rules 3 and 5 to 0.4, the normalized probabilities of the two parses are approx. 0.69 and 0.31 and the likelihood of the treebank is 0.147, which is higher than the likelihood with relative frequency estimates.

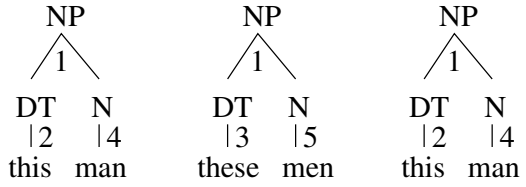

Figure 2: parse trees labeled with rules numbers

2.2 Log-Linear Models

The parameter estimation problem can be solved by turning the PCFG into a log-linear model. The features of the log-linear model are the grammar rules and the feature weights correspond to the logs of the rule probabilities. Equation 1 shows the relation between the models[3].

$$
\begin{aligned}
P(T) &= \frac{1}{Z}\prod_r p(r)^{f(r,T)} \qquad \text{(PCFG)} \\
&= \frac{1}{Z}\, e^{\sum_r f(r,T) log\, p(r)} \\
&= \frac{1}{Z}\, e^{\sum_i \nu_i(T)\, \lambda_i} \quad \text{(log-linear m.)}
\end{aligned}
$$

[3] Z is the normalization constant and $f(r,T)$ is the frequency of rule r in parse T.

Log-linear models are more general, however. They allow other properties than grammar rules and impose no restrictions on the weights. (Johnson et al., 1999) presented a computationally tractable gradient ascent training algorithm which maximizes the conditional probability of a parse given its yield. They also proposed a simulated annealing training algorithm which maximizes correct disambiguation rather than conditional probabilities.

Log-Linear models face an efficiency problem if the features are not locally defined. Grammar rules are local features, whereas features like `right-branching structure` or `parallel structure` and some other features used with log-linear models are not (see e.g. (Riezler et al., 2000)). Non-local features prevent the application of efficient dynamic programming methods for the computation of the most probable parses. Enumerating all analyses and scoring them individually, on the other hand, has a worst-case complexity which is exponential in the size of the parse forest and is therefore impractical for broad-coverage parsing of unrestricted text where sentences often have thousands of different analyses.

3 A Generative Probability Model

This section describes a new approach to probabilistic unification-based parsing which directly models a stochastic generator for UGs. The rule probabilities depend on the feature structure of the expanded constituent.

3.1 Generation with UGs

The generation of a parse tree consists of a sequence of rule applications. Each rule depends on the preceding rules. If e.g. the phrase `this man` is produced with the grammar given in fig. 1, the generator starts with an `NP` node and expands it to `DT N` using rule 1. According to the feature constraints, the `Number` features of the two daughter nodes are unified. `DT` is now expanded to `this` and the value of the `Number` feature becomes `sg`. Simultaneously, the `Number` feature of the `N` node gets the same value. In the next step, the generator can only choose a singular noun to expand the `N` node. So, the last action of the generator depends on the value of the `Number` feature. By assigning probabilities to the possible actions (i.e. grammar rules) of the generator, a probability model for parses is

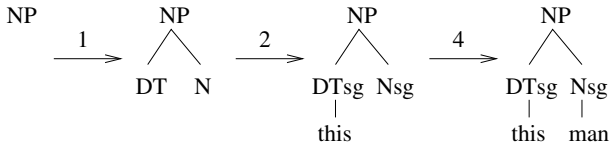

Figure 3: generation of the string *this man* (`DTsg` is an abbreviation for a node of category `DT` with the `Number` feature value `sg`.)

obtained. If only successful rules have a positive probability, the sum of all parse probabilities will be 1. Whether a rule is applicable or not depends on the feature structure of the expanded node. Therefore, the rule probabilities are conditioned on the expansion feature structures.

3.2 The Probability Model

For UGs with a unique start symbol[4], a parse tree is unambiguously defined by the sequence of grammar rules $r_1, \ldots, r_n$ and the generation order. Assuming some fixed generation order, the probability of a parse is given by $P(r_1, \ldots, r_n) = \prod_{i=1}^{n} p(r_i | r_1, \ldots, r_{i-1})$. Assuming that the probability of a rule depends only on the category c_i and the feature structure f_i of the expanded constituent, the probability of a parse is given by $P(T) = \prod_{i=1}^{n} p(r_i | c_i, f_i)$.[5]

According to this model, the probability $P(T)$ of the phrase `this man` is decomposed into the product of P(NP→DT N|NP), P(DT→this|DT) and P(N→man|Nsg). The relative frequency estimates obtained from the treebank in fig. 2 containing two parses for `this man` and one for `these men` are

$$\begin{aligned} P(NP \rightarrow DT\ N \mid NP) &= 1 \\ P(DT \rightarrow this \mid DT) &= 2/3 \\ P(DT \rightarrow these \mid DT) &= 1/3 \\ P(N \rightarrow man \mid Nsg) &= 1 \\ P(N \rightarrow men \mid Npl) &= 1 \end{aligned}$$

The probability of `this man` is therefore 2/3 and the probability of `these men` is 1/3. These probabilities are identical to the relative frequencies in the treebank and they sum to 1.

[4]For other grammars we add start symbol probabilities.

[5]The model puts rule sequences $r_1, \ldots, r_{i-1}$ yielding the same expansion context $\langle c_i, f_i \rangle$ into an equivalence class. Hence it might be considered as a special case of history-based parsing (see e.g. (Magerman, 1994)).

Note that the generation order has direct influence on the probability model. If the `N` node is expanded before the `DT` node, a different decomposition P(NP→DT N|NP) P(N→man|N) P(DT→this|DTsg) results. The `Number` feature of the noun is undefined when it is expanded whereas the `Number` feature of the determiner is `sg`.

Because of the influence of the generation order on the expansion context and therefore on the structure of the probability model, the generation order has to be chosen carefully. Assuming a head-first depth-first left-to-right generation order, the head of a rule should be defined such that the number of instantiated features is minimized, even if this definition diverges from a more linguistically motivated definition.

3.3 Computation of Parse Probabilities

The probability of a parse is the product of the rule probabilities given their expansion contexts. Expansion contexts are computed by running a generator which stores a copy of the current feature structure of each expanded node. Appendix A presents an algorithm for the efficient annotation of parse forests with expansion feature structures. Nodes with more than one expansion context are split. Annotated parse forests permit the computation of Inside, Outside and Viterbi probabilities with dynamic programming because expansion contexts are strictly local features. Algorithms for the computation of Viterbi and Inside-Outside probabilities are given e.g. in (Rooth and Schmid, 2001). Their time complexity is linear in the size of the parse forest.

3.4 Parameter Estimation

Parameters are either estimated with relative frequency estimates from treebanks or using the Inside-Outside algorithm on unlabeled data. However, there are three open questions: (i) The number of expansion contexts may be infinite. How are the parameters estimated? (ii) How are the parameters smoothed such that the probability of each consistent parse tree is positive? (iii) The generator may end up in a state where some non-terminal node cannot be expanded. How are such "dead end derivations" dealt with?

The first problem is solved by estimating parameters only for expansion contexts which show up in the training data. Rule probabilities

for other expansion contexts are estimated on the fly during parsing either by a uniform distribution over all matching grammar rules (i.e. rules unifiable with the feature structure of the expanded node) or by the averaged probability distribution (APD) of all expansion contexts with the same set of matching rules.

The second problem is solved by smoothing the rule probabilities such that every matching rule has a positive probability. Backoff smoothing with the APD distribution and a uniform distribution could be used here.

3.4.1 Unproductive Expansion Contexts

The third problem is the most difficult one. Not every expansion context is productive, i.e. is part of a successful derivation. If unproductive expansion contexts are generated with a positive probability, the probabilities of the consistent parses will not sum to 1 any more.

There are two types of unproductive expansion contexts. The first case is exemplified by the grammar in figure 4 which generates NP arguments and adds their feature structures to a subcat list sc[6]. The first rule can be used recur-

```
V {sc=Rest;} -> ‘V {sc=[X|Rest];} NP {}=X;
"v" : {sc=[NP{},NP{},NP{}];};
```

Figure 4: sample grammar with subcat features

sively again and again without causing a constraint violation, but the derivation will only succeed if a lexical rule with a matching subcat feature exists. A loss of probability mass to such dead end derivations is avoided by assigning zero probabilities to rules which generate nodes with unproductive expansion context. In the above example, the probability of the first rule should be zero for expansion contexts with three or more elements on the subcat list.

The grammar in figute 5 is an example for the second type of unproductive contexts. After applying the first rule and the third lexical rule, generation ends in a state where the `Number` feature of the `DT` node is instantiated with the value `pl`, and no rule matches anymore. Here it is not possible to assign a zero probability to the first

[6]The first rule unifies the feature structure of the NP node with the first element of the subcat list of the verb by means of the variable X. A subcat list consisting of all the other elements is passed on to the mother node by means of the variable *Rest*.

```
NP {} -> DT {Number=V;} ‘N {Number=V;};
NP {} -> ‘N {Number=V;};
"a" : DT {Number=sg;};
"book" : N {Number=sg;};
"books" : N {Number=pl;};
```

Figure 5: grammar causing dead ends of type II

lexical rule for the given expansion context because this rule is successfully applied in the same expansion context in a derivation starting with the second grammar rule. Whether the first lexical rule is successfully applicable or not depends on external information which is not represented in the local feature structure. The dead end disappears if the missing external information is added by modifying the first grammar rule as shown in fig. 6. The dead end also disappears if the generation order is changed such that the `DT` node is expanded first, or if a lexical rule for a plural determiner is added.

```
NP {} -> DT {Number=sg;} ‘N {Number=sg;};
```

Figure 6: modified NP rule

In order to obtain a probability distribution over parses, it is necessary to reduce the probability loss due to dead end derivation to 0. Dead ends of the first type are the smaller problem. Unsmoothed parameter estimates always assign a zero probability to such derivations because the problematic combinations of grammar rules and expansion contexts never appear in training data. If parameters are smoothed such that all matching rules have a positive probability, the probability of dead end derivations of the first type will decrease as the amount of training data increases approaching 0 asymptotically.

Dead ends of the second type always have a positive probability. The resulting probability loss can only be minimized by detecting and modifying the problematic rules.

3.4.2 Grammar Checking

A simple algorithm for detecting dead ends and for estimating (an upper bound of) the probability loss is the following:

CHECK_GRAMMAR(G, N)
1 $p_+ \leftarrow 0$
2 $p_- \leftarrow 0$
3 **for** N iterations

4 **do** create random derivation D with G
5 **if** D is a dead end derivation
6 **then** report a warning
7 $p_- \leftarrow p_- + p(D)$
8 **else** $p_+ \leftarrow p_+ + p(D)$

This algorithm detects dead end derivations of type II, and $\frac{p_-}{p_- + p_+}$ provides an estimate for the probability mass lost to dead end derivations of type II. $1 - p_+$ is an upper bound for the probability mass lost to all dead end derivations. This algorithm is not efficient, of course, and a chart-based algorithm should be used instead.

3.5 Comparison With PCFGs

If the number of expansion contexts is finite, it is possible to enumerate them and to detect all dead ends. Dead ends of type I are eliminated by assigning a zero probability to the problematic rule. Dead ends of type II have to be eliminated by modifying the grammar rules.

Grammars with a finite number of expansion contexts generate context-free languages and could be compiled into context-free grammars. So, the probabilistic UG could be replaced by a PCFG. We will show now with an example that the probabilistic UG has far fewer parameters than the PCFG.

Consider again the grammar in fig 4. If we add a lexicon entry for a transitive and a ditransitive verb and expand the V node before the NP node, the probability model for the UG has 4 different expansion contexts corresponding to a V node with 0, 1, 2 or 3 NPs on the subcat list. All NP features are undefined. There are two expansion contexts with one positive rule probability (V node with 0 or 3 NPs on the subcat list) and two contexts with two positive probabilities (V node with 1 or 2 NPs). Overall there are only two free parameters to be estimated.

Assuming that NPs have number (2 possible values), gender (3 values) and case features (4 values), the context-free grammar has 24 rules for V nodes with an empty subcat list, 24+24*24 rules for V nodes with one NP, 24*24+24*24*24 rules for V nodes with two NPs, and 24*24*24 rules for V nodes with three NPs. Altogether, there are 28249 rules and 13823 free parameters, which is a huge difference to the two free parameters of the probabilistic UG.

Is the probabilistic UG too simple? The model basically assigns probabilities to subcat lists of different length. However, in a more elaborated grammar, the probabilities of the verbal lexicon entries allow the model to distinguish how likely a particular subcat frame is for different verbs. Not captured by the model are dependencies between a verb and the agreement features of the arguments which are not explicitly defined in the lexicon entry. Lexical dependencies between words are also ignored in the model, but (Schmid, 2002) proposes a general method for the lexicalization of probabilistic grammars which could be used here.

3.6 Applicability

The presented model requires that a parse is unambiguously defined by a given sequence of grammar rules (assuming some fixed generation order). This poses problems if the grammar formalism is not rule-based (like HPSG) or if the result of a rule application is not uniquely defined (as with functional uncertainty in LFG). A compilation of such grammars into grammars in a more basic grammar formalism might be necessary in order to apply the model.

4 Comparison With Previous Work

Like PCFG approximations approaches, the presented model assigns probabilities exclusively to grammar rules, but the probabilities depend on the expansion contexts rather than on a fixed set of features compiled into a PCFG. Also, the number of parameters is smaller than in comparable PCFG and the training algorithm is consistent unless the grammar contains dead ends of type II (see section 3.4.2).

As in Goodman's model (1997), the probabilities depend on the previously instantiated features. However, Goodman's model assigns probabilities to feature values rather than grammar rules and the probabilities depend on fully instantiated feature structures.

5 Summary

A generative probability model for unification-based grammars was presented in which rule probabilities depend on the feature structures of the expanded nodes. The model requires no normalization and yields a probability distribution if all expansion contexts are productive. Efficient dynamic programming algorithms for disambiguation (Viterbi) and unsupervised train-

ing (Inside-Outside) are available and treebank training requires only relative frequency estimates. The training algorithms are consistent unless dead end derivations of a certain type appear. An algorithm was described which detects unproductive contexts and computes an upper bound for the probability mass lost to dead end derivations. The proposed model has a small number of parameters because many features are not instantiated in the expansion feature structures. In particular, it has fewer parameters than comparable PCFGs.

A Annotation with Expansion Context

A parse forest (see also Billot and Lang (1989)) in labeled grammar notation is a tuple $P = \langle N_P, \Sigma_P, R_P, S_P, M_P \rangle$ where $\langle N_P, \Sigma_P, R_P, S_P \rangle$ is a context free grammar (consisting of nonterminals N_P, terminals Σ_P, rules R_P, and start symbols S_P) and M_P is a function which maps elements of R_P to grammar rules in an underlying unification grammar. The grammar rules are represented as feature structures with multiple roots. The elements of N_P are pairs $\langle c, f \rangle$ consisting of a category c and a feature structure f. The elements of Σ_P are words. The functions *lhs* and *rhs* map rules to their left hand and right hand sides, respectively.

The following algorithm annotates a parse forest with expansion contexts and returns the annotated parse forest.

```
ANNOTATE(P)
 1 initialize P_T as an empty parse forest
 2 initialize P' as an empty parse forest
 3 initialize array L[N_P ∪ Σ_P] ← {}
 4 for v in S_P
 5     do PROCESS(v, ⊤, P, P_T, P', L, F)
 6 return P'
```

```
PROCESS(v, f_m, P, P_T, P', L, F)
 1 if ∃v' ∈ L[v] s.th. F[v'] = f_m
 2    then return
 3 if v ∈ Σ_P
 4    then v' ← NEWT(P', v)
 5         L[v] ← L[v] ∪ {v'}
 6         return
 7 v' ← NEWNT(P_T)
 8 L[v] ← L[v] ∪ {v'}
 9 for r in R_P s.th. lhs(r) = v
10     do f ← UNIFYM(F[v], M_P(r))
11        PROCESSD(r, v', ⟨⟩, f, P, P_T, P', L, F)
```

```
PROCESSD(r, v, d, f, P, P_T, P', L, F)
 1 l ← LENGTH(d) + 1 if l > LENGTH(rhs(r))
 2    then f_m ← MOTHERFS(f)
 3         if ∃v' ∈ L[v] s.th. F_e[v'] = f_m
 4            then ADD(r, v', d, P')
 5            else v' ← NEWNT(P')
 6                 L[v] ← L[v] ∪ {v'}
 7                 F_e[v'] = f_m
 8                 F[v'] = F[v]
 9                 ADD(r, v', d, P')
10    else
11         v_d ← Daughter(r, l);
12         f_d ← DaughterFS(f, n);
13         PROCESS(v_d, f_d, P, P_T, P', L, F, F_e)
14         for v'_d ∈ L[v_d]
15             do f' ← UNIFYD(F_e[v_d], f, l)
16                d' ← APPEND(d, v'_d)
17                PROCESSD(r, v, d', f')
```

The function ANNOTATE initializes data structures, calls the function PROCESS with each root node of the parse forest and returns the resulting annotated parse forest P'. L is used to link nodes to their annotated counterparts in another parse forest. The expansion feature structures are stored in F. F_e holds the feature structures of the nodes after the dominated subtree has been processed (post-expansion feature structures). Nodes annotated only with expansion feature structures but not with post-expansion feature structures are stored in the temporary parse forest P_T.

PROCESS annotates the subtree headed by the argument node. It checks, at first, whether it was called before with the same node and feature structure. If so, nothing has to be done. Otherwise PROCESS checks whether the argument node is a terminal node. If so, a new terminal node is created in the result parse forest and linked to the argument node. If the argument node is not a terminal node, a new nonterminal node is created in the temporary parse forest and linked to the argument node. PROCESS then iterates over all analyses of the argument node. It retrieves the grammar rule of the analysis (which is represented as a feature struc-

ture) and unifies the expansion feature structure of the argument node non-destructively with the mother feature structure of the grammar rule. The result is passed to the function ProcessD.

ProcessD checks whether all daughter nodes have been processed by comparing the length of the daughter list d (which is 0 in the first recursion) with the length of the right hand side of the rule. If both lengths are identical, the mother node feature structure is retrieved from the argument feature structure. If an equivalent node[7] has previously been inserted in the result parse tree, a new analysis with the daughter nodes listed in d is added to this node by calling the function `Add`. If no equivalent node exists, a new non-terminal v' is created and linked to the argument node v. The expansion feature structure and the post-expansion feature structure are stored in $F[v']$ and $F_e[v']$, respectively. Finally, the new analysis is added to v'.

If some daughter nodes are left for processing, ProcessD retrieves the next daughter node of the argument rule according to the generation order. It also retrieves the corresponding daughter feature structure from the argument feature structure and calls Process with these arguments. Afterwards, it iterates over all the annotated nodes which have been linked to the daughter node by Process. For each node, ProcessD unifies the post-expansion feature structure with the corresponding daughter node feature structure of the argument feature structure, appends the node to the list of daughter nodes and calls Process.

References

Steven Abney. 1997. Stochastic attribute-value grammars. *Computational Linguistics*, 23(4):597–618, December.

Sylvie Billot and Bernard Lang. 1989. The structure of shared forests in ambiguous parsing. In *Proceedings of the 27th Annual Meeting of the ACL, University of British Columbia*, Vancouver, B.C., Canada.

Chris Brew. 1995. Stochastic HPSG. In *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics*, Dublin.

Andreas Eisele. 1994. Towards probabilistic extensions of constraint-based grammars. In Jochen Dörre, editor, *Computational Aspects of constraint-based linguistic Description II*, pages 3–21. Institute for Computational Linguistics (IMS-CL), Stuttgart. DYANA-2 Deliverable R1.2.B.

Joshua Goodman. 1997. Probabilistic feature grammars. In *Proceedings of the 5th International Workshop on Parsing Technologies (IWPT '97)*, pages 89–100, Cambridge, Ma. Massachusetts Institute of Technology.

Mark Johnson, Stuart Geman, Stephen Canon, Chiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic "unification-based" grammars. In *Proceedings of the 37th Annual Meeting of the ACL*, College Park, MD.

Bernd Kiefer, Hans-Ulrich Krieger, and Detlef Prescher. 2002. A novel disambiguation method for unification-based grammars using probabilistic context-free approximations. internal report, Language Technology Lab, DFKI GmbH, Saarbrücken, Germany.

David M. Magerman. 1994. *Natural Language Processing as Statistical Pattern Recognition.* Ph.D. thesis, Stanford University.

Stefan Riezler, Detlef Prescher, Jonas Kuhn, and Mark Johnson. 2000. Lexicalized stochastic modeling of constraint-based grammars using log-linear measures and EM training. In *Proceedings of the 38th Annual Meeting of the ACL*, Hong Kong.

Mats Rooth and Helmut Schmid. 2001. Parse forest computation of expected governors. In *Proceedings of the 40th Annual Meeting of the ACL*, pages 458–465, Philadelphia, PA.

Helmut Schmid. 2000. *YAP: Parsing and Disambiguation With Feature-Based Grammars.* Ph.D. thesis, Institute for Computational Linguistics, University of Stuttgart, Germany.

Helmut Schmid. 2002. Lexicalization of probabilistic grammars. In *Proceedings of the 19th International Conference on Computational Linguistics*, Taipei, Taiwan.

Ronald M. Kaplan Richard Crouch John T. Maxwell III Stefan Riezler, Tracy H. King and Mark Johnson. 2002. Parsing the wall street journal using a lexical-functional grammar and discriminative estimation techniques.

[7] "Equivalent" means same feature structure, same expansion feature structure and same post-expansion feature structure.