# Wide-Coverage Efficient Statistical Parsing with CCG and Log-Linear Models

Stephen Clark*
University of Oxford

James R. Curran**
University of Sydney



*This article describes a number of log-linear parsing models for an automatically extracted lexicalized grammar. The models are "full" parsing models in the sense that probabilities are defined for complete parses, rather than for independent events derived by decomposing the parse tree. Discriminative training is used to estimate the models, which requires incorrect parses for each sentence in the training data as well as the correct parse. The lexicalized grammar formalism used is Combinatory Categorial Grammar (CCG), and the grammar is automatically extracted from CCGbank, a CCG version of the Penn Treebank. The combination of discriminative training and an automatically extracted grammar leads to a significant memory requirement (up to 25 GB), which is satisfied using a parallel implementation of the BFGS optimization algorithm running on a Beowulf cluster. Dynamic programming over a packed chart, in combination with the parallel implementation, allows us to solve one of the largest-scale estimation problems in the statistical parsing literature in under three hours.*

*A key component of the parsing system, for both training and testing, is a Maximum Entropy supertagger which assigns CCG lexical categories to words in a sentence. The supertagger makes the discriminative training feasible, and also leads to a highly efficient parser. Surprisingly, given CCG's "spurious ambiguity," the parsing speeds are significantly higher than those reported for comparable parsers in the literature. We also extend the existing parsing techniques for CCG by developing a new model and efficient parsing algorithm which exploits all derivations, including CCG's nonstandard derivations. This model and parsing algorithm, when combined with normal-form constraints, give state-of-the-art accuracy for the recovery of predicate–argument dependencies from CCGbank. The parser is also evaluated on DepBank and compared against the RASP parser, outperforming RASP overall and on the majority of relation types. The evaluation on DepBank raises a number of issues regarding parser evaluation.*

*This article provides a comprehensive blueprint for building a wide-coverage CCG parser. We demonstrate that both accurate and highly efficient parsing is possible with CCG.*

* University of Oxford Computing Laboratory, Wolfson Building, Parks Road, Oxford, OX1 3QD, UK. E-mail: stephen.clark@comlab.ox.ac.uk.
** School of Information Technologies, University of Sydney, NSW 2006, Australia. E-mail: james@it.usyd.edu.au.

Submission received: 27 April 2006; revised submission received: 30 November 2006; accepted for publication: 16 March 2007.

## 1. Introduction

Log-linear models have been applied to a number of problems in NLP, for example, POS tagging (Ratnaparkhi 1996; Lafferty, McCallum, and Pereira 2001), named entity recognition (Borthwick 1999), chunking (Koeling 2000), and parsing (Johnson et al. 1999). Log-linear models are also referred to as **maximum entropy models** and **random fields** in the NLP literature. They are popular because of the ease with which complex discriminating features can be included in the model, and have been shown to give good performance across a range of NLP tasks.

Log-linear models have previously been applied to statistical parsing (Johnson et al. 1999; Toutanova et al. 2002; Riezler et al. 2002; Malouf and van Noord 2004), but typically under the assumption that all possible parses for a sentence can be enumerated. For manually constructed grammars, this assumption is usually sufficient for efficient estimation and decoding. However, for wide-coverage grammars extracted from a treebank, enumerating all parses is infeasible. In this article we apply the dynamic programming method of Miyao and Tsujii (2002) to a packed chart; however, because the grammar is automatically extracted, the packed charts require a considerable amount of memory: up to 25 GB. We solve this massive estimation problem by developing a parallelized version of the estimation algorithm which runs on a Beowulf cluster.

The lexicalized grammar formalism we use is Combinatory Categorial Grammar (CCG; Steedman 2000). A number of statistical parsing models have recently been developed for CCG and used in parsers applied to newspaper text (Clark, Hockenmaier, and Steedman 2002; Hockenmaier and Steedman 2002b; Hockenmaier 2003b). In this article we extend existing parsing techniques by developing log-linear models for CCG, as well as a new model and efficient parsing algorithm which exploits all CCG's derivations, including the nonstandard ones.

Estimating a log-linear model involves computing expectations of feature values. For the conditional log-linear models used in this article, computing expectations requires a sum over all derivations for each sentence in the training data. Because there can be a massive number of derivations for some sentences, enumerating all derivations is infeasible. To solve this problem, we have adapted the dynamic programming method of Miyao and Tsujii (2002) to packed CCG charts. A packed chart efficiently represents all derivations for a sentence. The dynamic programming method uses inside and outside scores to calculate expectations, similar to the inside–outside algorithm for estimating the parameters of a PCFG from unlabeled data (Lari and Young 1990).

Generalized Iterative Scaling (Darroch and Ratcliff 1972) is a common choice in the NLP literature for estimating a log-linear model (e.g., Ratnaparkhi 1998; Curran and Clark 2003). Initially we used generalized iterative scaling (GIS) for the parsing models described here, but found that convergence was extremely slow; Sha and Pereira (2003) present a similar finding for globally optimized log-linear models for sequences. As an alternative to GIS, we use the limited-memory BFGS algorithm (Nocedal and Wright 1999). As Malouf (2002) demonstrates, general purpose numerical optimization algorithms such as BFGS can converge much faster than iterative scaling algorithms (including Improved Iterative Scaling; Della Pietra, Della Pietra, and Lafferty 1997).

Despite the use of a packed representation, the complete set of derivations for the sentences in the training data requires up to 25 GB of RAM for some of the models in this article. There are a number of ways to solve this problem. Possibilities include using a subset of the training data; repeatedly parsing the training data for each iteration of the estimation algorithm; or reading the packed charts from disk for each iteration.

These methods are either too slow or sacrifice parsing performance, and so we use a parallelized version of BFGS running on an 18-node Beowulf cluster to perform the estimation. Even given the large number of derivations and the large feature sets in our models, the estimation time for the best-performing model is less than three hours. This gives us a practical framework for developing a statistical parser.

A corollary of CCG's base-generative treatment of long-range dependencies in relative clauses and coordinate constructions is that the standard predicate–argument relations can be derived via nonstandard surface derivations. The addition of "spurious" derivations in CCG complicates the modeling and parsing problems. In this article we consider two solutions. The first, following Hockenmaier (2003a), is to define a model in terms of **normal-form** derivations (Eisner 1996). In this approach we recover only one derivation leading to a given set of predicate–argument dependencies and ignore the rest.

The second approach is to define a model over the predicate–argument dependencies themselves, by summing the probabilities of all derivations leading to a given set of dependencies. We also define a new efficient parsing algorithm for such a model, based on Goodman (1996), which maximizes the expected recall of dependencies. The development of this model allows us to test, for the purpose of selecting the correct predicate–argument dependencies, whether there is useful information in the additional derivations. We also compare the performance of our best log-linear model against existing CCG parsers, obtaining the highest results to date for the recovery of predicate–argument dependencies from CCGbank.

A key component of the parsing system is a Maximum Entropy CCG supertagger (Ratnaparkhi 1996; Curran and Clark 2003) which assigns lexical categories to words in a sentence. The role of the supertagger is twofold. First, it makes discriminative estimation feasible by limiting the number of incorrect derivations for each training sentence; the supertagger can be thought of as supplying a number of incorrect but plausible lexical categories for each word in the sentence. Second, it greatly increases the efficiency of the parser, which was the original motivation for supertagging (Bangalore and Joshi 1999). One possible criticism of CCG has been that highly efficient parsing is not possible because of the additional "spurious" derivations. In fact, we show that a novel method which tightly integrates the supertagger and parser leads to parse times significantly faster than those reported for comparable parsers in the literature.

The parser is evaluated on CCGbank (available through the Linguistic Data Consortium). In order to facilitate comparisons with parsers using different formalisms, we also evaluate on the publicly available DepBank (King et al. 2003), using the Briscoe and Carroll annotation consistent with the RASP parser (Briscoe, Carroll, and Watson 2006). The dependency annotation is designed to be as theory-neutral as possible to allow easy comparison. However, there are still considerable difficulties associated with a cross-formalism comparison, which we describe. Even though the CCG dependencies are being mapped into another representation, the accuracy of the CCG parser is over 81% F-score on labeled dependencies, against an upper bound of 84.8%. The CCG parser also outperforms RASP overall and on the majority of dependency types.

The contributions of this article are as follows. First, we explain how to estimate a full log-linear parsing model for an automatically extracted grammar, on a scale as large as that reported anywhere in the NLP literature. Second, the article provides a comprehensive blueprint for building a wide-coverage CCG parser, including theoretical and practical aspects of the grammar, the estimation process, and decoding. Third, we investigate the difficulties associated with cross-formalism parser comparison, evaluating the parser on DepBank. And finally, we develop new models and decoding algorithms for

CCG, and give a convincing demonstration that, through use of a supertagger, highly efficient parsing is possible with CCG.

## 2. Related Work

The first application of log-linear models to parsing is the work of Ratnaparkhi and colleagues (Ratnaparkhi, Roukos, and Ward 1994; Ratnaparkhi 1996, 1999). Similar to Della Pietra, Della Pietra, and Lafferty (1997), Ratnaparkhi motivates log-linear models from the perspective of maximizing entropy, subject to certain constraints. Ratnaparkhi models the various decisions made by a shift-reduce parser, using log-linear distributions defined over features of the local context in which a decision is made. The probabilities of each decision are multiplied together to give a score for the complete sequence of decisions, and beam search is used to find the most probable sequence, which corresponds to the most probable derivation.

A different approach is proposed by Abney (1997), who develops log-linear models for attribute-value grammars, such as Head-driven Phrase Structure Grammar (HPSG). Rather than define a model in terms of parser moves, Abney defines a model directly over the syntactic structures licensed by the grammar. Another difference is that Abney uses a global model, in which a single log-linear model is defined over the complete space of attribute–value structures. Abney's motivation for using log-linear models is to overcome various problems in applying models based on PCFGs directly to attribute-value grammars. A further motivation for using global models is that these do not suffer from the label bias problem (Lafferty, McCallum, and Pereira 2001), which is a potential problem for Ratnaparkhi's approach.

Abney defines the following model for a syntactic analysis $\omega$:

$$P(\omega) = \frac{\prod_i \beta_i^{f_i(\omega)}}{Z} \tag{1}$$

where $f_i(\omega)$ is a feature, or feature function, and $\beta_i$ is its corresponding weight; $Z$ is a normalizing constant, also known as the **partition function**. In much work using log-linear models in NLP, including Ratnaparkhi's, the features of a model are indicator functions which take the value 0 or 1. However, in Abney's models, and in the models used in this article, the feature functions are integer valued and count the number of times some feature appears in a syntactic analysis.[1] Abney calls the feature functions **frequency functions** and, like Abney, we will not always distinguish between a feature and its corresponding frequency function.

There are practical difficulties with Abney's proposal, in that finding the maximum-likelihood solution during estimation involves calculating expectations of feature values, which are sums over the complete space of possible analyses. Abney suggests a Metropolis-Hastings sampling procedure for calculating the expectations, but does not experiment with an implementation.

Johnson et al. (1999) propose an alternative solution, which is to maximize the conditional likelihood function. In this case the likelihood function is the product of the *conditional* probabilities of the syntactic analyses in the data, each probability conditioned on the respective sentence. The advantage of this method is that calculating the conditional feature expectations only requires a sum over the syntactic analyses for the

1 In principle the features could be real-valued, but we only use integer-valued features in this article.

sentences in the training data. The conditional-likelihood estimator is also consistent for the conditional distributions (Johnson et al. 1999). The same solution is arrived at by Della Pietra, Della Pietra, and Lafferty (1997) via a maximum entropy argument. Another feature of Johnson et al.'s approach is the use of a Gaussian prior term to avoid overfitting, which involves adding a regularization term to the likelihood function; the regularization term penalizes models whose weights get too large in absolute value. This smoothing method for log-linear models is also proposed by Chen and Rosenfeld (1999).

Calculating the conditional feature expectations can still be problematic if the grammar licenses a large number of analyses for some sentences. This is not a problem for Johnson et al. (1999) because their grammars are hand-written and constraining enough to allow the analyses for each sentence to be enumerated. However, for grammars with wider coverage it is often not possible to enumerate the analyses for each sentence in the training data. Osborne (2000) investigates training on a sample of the analyses for each sentence, for example the top-$n$ most probable according to some other probability model, or simply a random sample.

The CCG grammar used in this article is automatically extracted, has wide coverage, and can produce an extremely large number of derivations for some sentences, far too many to enumerate. We adapt the **feature-forest** method of Miyao and Tsujii (2002), which involves using dynamic programming to efficiently calculate the feature expectations. Geman and Johnson (2002) propose a similar method in the context of LFG parsing; an implementation is described in Kaplan et al. (2004).

Miyao and Tsujii have carried out a number of investigations similar to the work in this article. In Miyao and Tsujii (2003b, 2003a) log-linear models are developed for automatically extracted grammars for Lexicalized Tree Adjoining Grammar (LTAG) and Head Driven Phrase Structure Grammar (HPSG). One of Miyao and Tsujii's motivations is to model predicate–argument dependencies, including long-range dependencies, which was one of the original motivations of the wide-coverage CCG parsing project. Miyao and Tsujii (2003a) present another log-linear model for an automatically extracted LTAG which uses a simple unigram model of the elementary trees together with a log-linear model of the attachments. Miyao and Tsujii (2005) address the issue of practical estimation using an automatically extracted HPSG grammar. A simple unigram model of lexical categories is used to limit the size of the charts for training, in a similar way to how we use a CCG supertagger to restrict the size of the charts.

The main differences between Miyao and Tsujii's work and ours, aside from the different grammar formalisms, are as follows. The CCG supertagger is a key component of our parsing system. It allows practical estimation of the log-linear models as well as highly efficient parsing. The Maximum Entropy supertagger we use could also be applied to Miyao and Tsujii's grammars, although whether similar performance would be obtained depends on the characteristics of the grammar; see subsequent sections for more discussion of this issue in relation to LTAG. The second major difference is in our use of a cluster and parallelized estimation algorithm. We have found that significantly increasing the size of the parse space available for discriminative estimation, which is possible on the cluster, improves the accuracy of the resulting parser. Another advantage of parallelization, as discussed in Section 5.5, is the reduction in estimation time. Again, our parallelization techniques could be applied to Miyao and Tsujii's framework.

Malouf and van Noord (2004) present similar work to ours, in the context of an HPSG grammar for Dutch. One similarity is that their parsing system uses an HMM tagger before parsing, similar to our supertagger. One difference is that we use a Maximum Entropy tagger which allows more flexibility in terms of the features that can

be encoded; for example, we have found that using Penn Treebank POS tags as features significantly improves supertagging accuracy. Another difference is that Malouf and van Noord use the random sampling method of Osborne (2000) to allow practical estimation, whereas we construct the complete parse forest but use the supertagger to limit the size of the charts. Their work is also on a somewhat smaller scale, with the Dutch Alpino treebank containing 7,100 sentences, compared with the 36,000 sentences we use for training.

Kaplan et al. (2004) present similar work to ours in the context of an LFG grammar for English. The main difference is that the LFG grammar is hand-built, resulting in less ambiguity than an automatically extracted grammar and thus requiring fewer resources for model estimation. One downside of hand-built grammars is that they are typically less robust, which Kaplan et al. address by developing a "fragment" grammar, together with a "skimming mode," which increases coverage on Section 23 of the Penn Treebank from 80% to 100%. Kaplan et al. also present speed figures for their parser, comparing with the Collins parser. Comparing parser speeds is difficult because of implementation and accuracy differences, but their highest reported speed is around 2 sentences per second on sentences from Section 23. The parse speeds that we present in Section 10.3 are an order of magnitude higher.

More generally, the literature on statistical parsing using linguistically motivated grammar formalisms is large and growing. Statistical parsers have been developed for TAG (Chiang 2000; Sarkar and Joshi 2003), LFG (Riezler et al. 2002; Kaplan et al. 2004; Cahill et al. 2004), and HPSG (Toutanova et al. 2002; Toutanova, Markova, and Manning 2004; Miyao and Tsujii 2004; Malouf and van Noord 2004), among others. The motivation for using these formalisms is that many NLP tasks, such as Machine Translation, Information Extraction, and Question Answering, could benefit from the more sophisticated linguistic analyses they provide.

The formalism most closely related to CCG from this list is TAG. TAG grammars have been automatically extracted from the Penn Treebank, using techniques similar to those used by Hockenmaier (Chen and Vijay-Shanker 2000; Xia, Palmer, and Joshi 2000). Also, the supertagging idea which is central to the efficiency of the CCG parser originated with TAG (Bangalore and Joshi 1999). Chen et al. (2002) describe the results of reranking the output of an HMM supertagger using an automatically extracted LTAG. The accuracy for a single supertag per word is slightly over 80%. This figure is increased to over 91% when the tagger is run in *n*-best mode, but at a considerable cost in ambiguity, with 8 supertags per word. Nasr and Rambow (2004) investigate the potential impact of LTAG supertagging on parsing speed and accuracy by performing a number of oracle experiments. They find that, *with the perfect supertagger*, extremely high parsing accuracies and speeds can be obtained. Interestingly, the accuracy of LTAG supertaggers using automatically extracted grammars is significantly below the accuracy of the CCG supertagger. One possible way to increase the accuracy of LTAG supertagging is to use a Maximum Entropy, rather than HMM, tagger (as discussed previously), but this is likely to result in an improvement of only a few percentage points. Thus whether the difference in supertagging accuracy is due to the nature of the formalisms, the supertagging methods used, or properties of the extracted grammars, is an open question.

Related work on statistical parsing with CCG will be described in Section 3.

## 3. Combinatory Categorial Grammar

Combinatory Categorial Grammar (CCG) (Steedman 1996, 2000) is a type-driven lexicalized theory of grammar based on Categorial Grammar (Wood 1993). CCG lexical

entries consist of a syntactic category, which defines valency and directionality, and a semantic interpretation. In this article we are concerned with the syntactic component; see Steedman (2000) for how a semantic interpretation can be composed during a syntactic derivation, and also Bos et al. (2004) for how semantic interpretations can be built for newspaper text using the wide-coverage parser described in this article.

Categories can be either basic or complex. Examples of basic categories are $S$ (sentence), $N$ (noun), $NP$ (noun phrase), and $PP$ (prepositional phrase). Complex categories are built recursively from basic categories, and indicate the type and directionality of arguments (using slashes), and the type of the result. For example, the following category for the transitive verb *bought* specifies its first argument as a noun phrase to its right, its second argument as a noun phrase to its left, and its result as a sentence:

$$\text{bought} := (S\backslash NP)/NP \tag{2}$$

In the theory of CCG, basic categories are regarded as complex objects that include syntactic features such as number, gender, and case. For the grammars in this article, categories are augmented with some additional information, such as head information, and also features on $S$ categories which distinguish different types of sentence, such as declarative, infinitival, and *wh*-question. This additional information will be described in later sections.

Categories are combined in a derivation using **combinatory rules**. In the original Categorial Grammar (Bar-Hillel 1953), which is context-free, there are two rules of **functional application**:

$$X/Y \;\; Y \Rightarrow X \quad (>) \tag{3}$$

$$Y \;\; X\backslash Y \Rightarrow X \quad (<) \tag{4}$$

where $X$ and $Y$ denote categories (either basic or complex). The first rule is **forward application** ($>$) and the second rule is **backward application** ($<$). Figure 1 gives an example derivation using these rules.

CCG extends the original Categorial Grammar by introducing a number of additional combinatory rules. The first is **forward composition**, which Steedman denotes

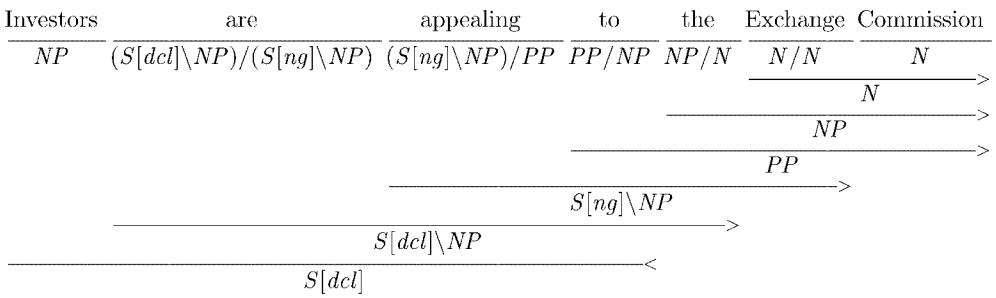



**Figure 1**
Example derivation using forward and backward application.

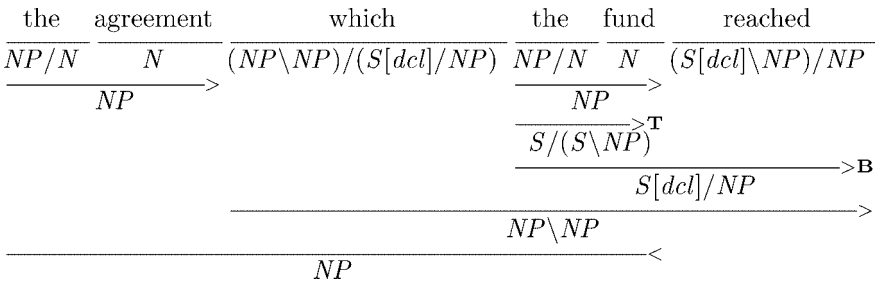



**Figure 2**
Example derivation using type-raising and forward composition.



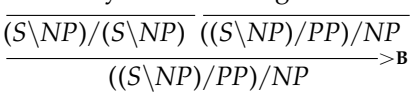
by $>$ **B** (because **B** is the symbol used by Curry to denote function composition in combinatory logic; Curry and Feys 1958):

$$X/Y \;\; Y/Z \quad \Rightarrow_{\mathbf{B}} \quad X/Z \quad (>\mathbf{B}) \tag{5}$$

Forward composition is often used in conjunction with **type-raising** (**T**), as in Figure 2. In this case type-raising takes a subject noun phrase and turns it into a functor looking to the right for a verb phrase; *the fund* is then able to combine with *reached* using forward composition, giving *the fund reached* the category $S[dcl]/NP$ (a declarative sentence missing an object). It is exactly this type of constituent which the object relative pronoun category is looking for to its right: $(NP\backslash NP)/(S[dcl]/NP)$.

Note that *the fund reached* is a perfectly reasonable constituent in CCG, having the type $S[dcl]/NP$. This allows analyses for sentences such as *the fund reached but investors disagreed with the agreement*, even though this construction is often described as "non-constituent coordination." In this example, *the fund reached* and *investors disagreed with* have the same type, allowing them to be coordinated, resulting in *the fund reached but investors disagreed with* having the type $S[dcl]/NP$. Note also that it is this flexible notion of constituency which leads to so-called spurious ambiguity, because even the simple sentence *the fund reached an agreement* will have more than one derivation, with each derivation leading to the same set of predicate–argument dependencies.

Forward composition is generalized to allow additional arguments to the right of the $Z$ category in (5). For example, the following combination allows analysis of sentences such as *I offered, and may give, a flower to a policeman* (Steedman 2000):




This example shows how the categories for *may* and *give* combine, resulting in a category of the same type as *offered*, which can then be coordinated. Steedman (2000) gives a more precise definition of generalized forward composition.

Further combinatory rules in the theory of CCG include backward composition ($<$ **B**) and backward crossed composition ($<$ $\mathbf{B}_{\times}$):

$$Y\backslash Z \;\; X\backslash Y \Rightarrow_{\mathbf{B}} X\backslash Z \quad (<\mathbf{B}) \tag{6}$$

$$Y/Z \;\; X\backslash Y \Rightarrow_{\mathbf{B}} X/Z \quad (<\mathbf{B}_{\times}) \tag{7}$$

Backward composition provides an analysis for sentences involving "argument cluster coordination," such as *I gave a teacher an apple and a policeman a flower* (Steedman 2000). Backward crossed composition is required for heavy NP shift and coordinations such as *I shall buy today and cook tomorrow the mushrooms*. In this coordination example from Steedman (2000), backward crossed composition is used to combine the categories for *buy*, $(S\backslash NP)/NP$, and *today*, $(S\backslash NP)\backslash(S\backslash NP)$, and similarly for *cook* and *tomorrow*, producing categories of the same type which can be coordinated. This rule is also generalized in an analogous way to forward composition.

Finally, there is a coordination rule which conjoins categories of the same type, producing a further category of that type. This rule can be implemented by assuming the following category schema for a coordination term: $(X\backslash X)/X$, where $X$ can be any category.

All of the combinatory rules described above are implemented in our parser. Other combinatory rules, such as substitution, have been suggested in the literature to deal with certain linguistic phenomena, but we chose not to implement them. The reason is that adding new combinatory rules reduces the efficiency of the parser, and we felt that, in the case of substitution, for example, the small gain in grammatical coverage was not worth the reduction in speed. Section 9.3 discusses some of the choices we made when implementing the grammar.

One way of dealing with the additional ambiguity in CCG is to only consider **normal-form** derivations. Informally, a normal-form derivation is one which uses type-raising and composition only when necessary. Eisner (1996) describes a technique for eliminating spurious ambiguity entirely, by defining exactly one normal-form derivation for each semantic equivalence class of derivations. The idea is to restrict the combination of categories produced by composition; more specifically, any constituent which is the result of a forward composition cannot serve as the primary (left) functor in another forward composition or forward application. Similarly, any constituent which is the result of a backward composition cannot serve as the primary (right) functor in another backward composition or backward application. Eisner only deals with a grammar without type-raising, and so the constraints cannot guarantee a normal-form derivation when applied to the grammars used in this article. However, the constraints can still be used to significantly reduce the parsing space. Section 9.3 describes the various normal-form constraints used in our experiments.

A recent development in the theory of CCG is the **multi-modal** treatment given by Baldridge (2002) and Baldridge and Kruijff (2003), following the type-logical approaches to Categorial Grammar (Moortgat 1997). One possible extension to the parser and grammar described in this article is to incorporate the multi-modal approach; Baldridge suggests that, as well as having theoretical motivation, a multi-modal approach can improve the efficiency of CCG parsing.

### 3.1 Why Use CCG for Statistical Parsing?

CCG was designed to deal with the long-range dependencies inherent in certain constructions, such as coordination and extraction, and arguably provides the most linguistically satisfactory account of these phenomena. Long-range dependencies are relatively common in text such as newspaper text, but are typically not recovered by treebank parsers such as Collins (2003) and Charniak (2000). This has led to a number of proposals for post-processing the output of the Collins and Charniak parsers, in which trace sites are located and the antecedent of the trace determined (Johnson 2002; Dienes and Dubey 2003; Levy and Manning 2004). An advantage of using CCG is that

the recovery of long-range dependencies can be integrated into the parsing process in a straightforward manner, rather than be relegated to such a post-processing phase (Clark, Hockenmaier, and Steedman 2002; Hockenmaier 2003a; Clark, Steedman, and Curran 2004).

Another advantage of CCG is that providing a compositional semantics for the grammar is relatively straightforward. It has a completely transparent interface between syntax and semantics and, because CCG is a lexicalized grammar formalism, providing a compositional semantics simply involves adding semantic representations to the lexical entries and interpreting the small number of combinatory rules. Bos et al. (2004) show how this can be done for the grammar and parser described in this article.

Of course some of these advantages could be obtained with other grammar formalisms, such as TAG, LFG, and HPSG, although CCG is especially well-suited to analysing coordination and long-range dependencies. For example, the analysis of "non-constituent coordination" described in the previous section is, as far as we know, unique to CCG.

Finally, the lexicalized nature of CCG has implications for the engineering of a wide-coverage parser. Later we show that use of a *supertagger* (Bangalore and Joshi 1999) prior to parsing can produce an extremely efficient parser. The supertagger uses statistical sequence tagging techniques to assign a small number of lexical categories to each word in the sentence. Because there is so much syntactic information in lexical categories, the parser is required to do less work once the lexical categories have been assigned; hence Srinivas and Joshi, in the context of TAG, refer to supertagging as *almost parsing*. The parser is able to parse 20 *Wall Street Journal* (WSJ) sentences per second on standard hardware, using our best-performing model, which compares very favorably with other parsers using linguistically motivated grammars.

A further advantage of the supertagger is that it can be used to reduce the parse space for estimation of the log-linear parsing models. By focusing on those parses which result from the most probable lexical category sequences, we are able to perform effective discriminative training without considering the complete parse space, which for most sentences is prohibitively large.

The idea of supertagging originated with LTAG; however, in contrast to the CCG grammars used in this article, the automatically extracted LTAG grammars have, as yet, been too large to enable effective supertagging (as discussed in the previous section). We are not aware of any other work which has demonstrated the parsing efficiency benefits of supertagging using an automatically extracted grammar.

### 3.2 Previous Work on CCG Statistical Parsing

The work in this article began as part of the Edinburgh wide-coverage CCG parsing project (2000–2004). There has been some other work on defining stochastic categorial grammars, but mainly in the context of grammar learning (Osborne and Briscoe 1997; Watkinson and Manandhar 2001; Zettlemoyer and Collins 2005).

An early attempt from the Edinburgh project at wide-coverage CCG parsing is presented in Clark, Hockenmaier, and Steedman (2002). In order to deal with the problem of the additional, nonstandard CCG derivations, a conditional model of dependency structures is presented, based on Collins (1996), in which the dependencies are modeled directly and derivations are not modeled at all. The conditional probability of a dependency structure $\pi$, given a sentence $S$, is factored into two parts. The first part is the probability of the lexical category sequence, $C$, and the second part is the dependency structure, $D$, giving $P(\pi|S) = P(C|S)P(D|C, S)$. Intuitively, the category sequence is gen-

erated first, conditioned on the sentence, and then attachment decisions are made to form the dependency links. The probability of the category sequence is estimated using a maximum entropy model, following the supertagger described in Clark (2002). The probabilities of the dependencies are estimated using relative frequencies, following Collins (1996).

The model was designed to include some long-range predicate–argument dependencies, as well as local dependencies. However, there are a number of problems with the model, as the authors acknowledge. First, the model is deficient, losing probability mass to dependency structures not generated by the grammar. Second, the relative frequency estimation of the dependency probabilities is ad hoc, and cannot be seen as maximum likelihood estimation, or some other principled method. Despite these flaws, the parser based on this model was able to recover CCG predicate–argument dependencies at around 82% overall F-score on unseen WSJ text.

Hockenmaier (2003a) and Hockenmaier and Steedman (2002b) present a generative model of normal-form derivations, based on various techniques from the statistical parsing literature (Charniak 1997; Goodman 1997; Collins 2003). A CCG binary derivation tree is generated top-down, with the probability of generating particular child nodes being conditioned on some limited context from the previously generated structure. Hockenmaier's parser uses rule instantiations read off CCGbank (see Section 3.3) and some of these will be instances of type-raising and composition; hence the parser can produce non-normal-form derivations. However, because the parsing model is estimated over normal-form derivations, any non-normal-form derivations will receive low probabilities and are unlikely to be returned as the most probable parse.

Hockenmaier (2003a) compares a number of generative models, starting with a baseline model based on a PCFG. Various extensions to the baseline are considered: increasing the amount of lexicalization; generating a lexical category at its maximal projection; conditioning the probability of a rule instantiation on the grandparent node (Johnson 1998); adding features designed to deal with coordination; and adding distance to the dependency features. Some of these extensions, such as increased lexicalization and generating a lexical category at its maximal projection, improved performance, whereas others, such as the coordination and distance features, reduced performance. Hockenmaier (2003a) conjectures that the reduced performance is due to the problem of data sparseness, which becomes particularly severe for the generative model when the number of features is increased. The best performing model outperforms that of Clark, Hockenmaier, and Steedman (2002), recovering CCG predicate–argument dependencies with an overall F-score of around 84% using a similar evaluation.

Hockenmaier (2003b) presents another generative model of normal-form derivations, which is based on the dependencies in the predicate–argument structure, including long-range dependencies, rather than the dependencies defined by the local trees in the derivation. Hockenmaier also argues that, compared to Hockenmaier and Steedman (2002b), the predicate–argument model is better suited to languages with freer word order than English. The model was also designed to test whether the inclusion of predicate–argument dependencies improves parsing accuracy. In fact, the results given in Hockenmaier (2003b) are lower than previous results. However, Hockenmaier (2003b) reports that the increased complexity of the model reduces the effectiveness of the dynamic programming used in the parser, and hence a more aggressive beam search is required to produce reasonable parse times. Thus the reduced accuracy could be due to implementation difficulties rather than the model itself.

The use of conditional log-linear models in this article is designed to overcome some of the weaknesses identified in the approach of Clark, Hockenmaier, and Steedman

(2002), and to offer a more flexible framework for including features than the generative models of Hockenmaier (2003a). For example, adding long-range dependency features to the log-linear model is straightforward. We also showed in Clark and Curran (2004b) that, in contrast with Hockenmaier (2003a), adding distance to the dependency features in the log-linear model does improve parsing accuracy. Another feature of conditional log-linear models is that they are trained discriminatively, by maximizing the conditional probability of each gold-standard parse relative to the incorrect parses for the sentence. Generative models, in contrast, are typically trained by maximizing the *joint* probability of the ⟨training sentence, parse⟩ pairs, even though the sentence does not need to be inferred.

### 3.3 CCGbank

The treebank used in this article performs two roles: It provides the lexical category set used by the supertagger, plus some unary type-changing rules and punctuation rules used by the parser, and it is used as training data for the statistical models. The treebank is CCGbank (Hockenmaier and Steedman 2002a; Hockenmaier 2003a), a CCG version of the Penn Treebank (Marcus, Santorini, and Marcinkiewicz 1993). Penn Treebank conversions have also been carried out for other linguistic formalisms, including TAG (Chen and Vijay-Shanker 2000; Xia, Palmer, and Joshi 2000), LFG (Burke et al. 2004), and HPSG (Miyao, Ninomiya, and Tsujii 2004).

CCGbank was created by converting the phrase-structure trees in the Penn Treebank into CCG normal-form derivations. Some preprocessing of the phrase-structure trees was required, in order to allow the correct CCG analyses for some constructions, such as coordination. Hockenmaier (2003a) gives a detailed description of the procedure used to create CCGbank. Figure 3 shows an example normal-form derivation for an (abbreviated) CCGbank sentence. The derivation has been inverted, so that it is represented as a binary tree.

Sentence categories (*S*) in CCGbank carry features, such as [*dcl*] for declarative, [*wq*] for *wh*-questions, and [*for*] for small clauses headed by *for*; see Hockenmaier (2003a) for the complete list. *S* categories also carry features in verb phrases; for example, $S[b]\backslash NP$

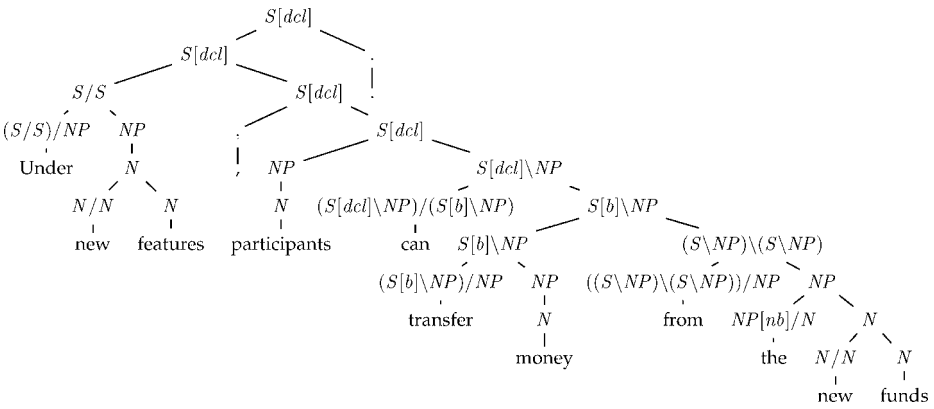



**Figure 3**
Example CCG derivation as a binary tree for the sentence *Under new features, participants can transfer money from the new funds.*

is a bare-infinitive; $S[to]\backslash NP$ is a *to*-infinitive; $S[pss]\backslash NP$ is a past participle in passive mode. Note that, whenever an $S$ or $S\backslash NP$ category is modified, any feature on the $S$ is carried through to the result category; this is true in our parser also. Finally, determiners specify that the resulting noun phrase is non-bare: $NP[nb]/N$, although this feature is largely ignored by the parser described in this article.

As well as instances of the standard CCG combinatory rules—forward and backward application, forward and backward composition, backward-crossed composition, type-raising, coordination of like types—CCGbank contains a number of unary **type-changing** rules and rules for dealing with punctuation. The type-changing rules typically change a verb phrase into a modifier. The following examples, taken from Hockenmaier (2003a), demonstrate the most common rules. The bracketed expression has the type-changing rule applied to it:

- $S[pss]\backslash NP \Rightarrow NP\backslash NP$
  *workers [exposed to it]*
- $S[adj]\backslash NP \Rightarrow NP\backslash NP$
  *a forum [likely to bring attention to the problem]*
- $S[ng]\backslash NP \Rightarrow NP\backslash NP$
  *signboards [advertising imported cigarettes]*
- $S[ng]\backslash NP \Rightarrow (S\backslash NP)\backslash(S\backslash NP)$
  *became chairman [succeeding Ian Butler]*
- $S[dcl]/NP \Rightarrow NP\backslash NP$
  *the millions of dollars [it generates]*

Another common type-changing rule in CCGbank, which appears in Figure 3, changes a noun category $N$ into a noun phrase $NP$. Appendix A lists the unary type-changing rules used by our parser.

There are also a number of rules in CCGbank for absorbing punctuation. For example, Figure 3 contains a rule which takes a comma followed by a declarative sentence and returns a declarative sentence:

$$, \quad S[dcl] \Rightarrow S[dcl]$$

There are a number of similar comma rules for other categories. There are also similar punctuation rules for semicolons, colons, and brackets. There is also a rule schema which treats a comma as a coordination:

$$, \quad X \Rightarrow X\backslash X$$

Appendix A contains the complete list of punctuation rules used in the parser.

A small number of local trees in CCGbank—consisting of a parent and one or two children—do not correspond to any of the CCG combinatory rules, or the type-changing rules or punctuation rules. This is because some of the phrase structure subtrees in the

Penn Treebank are difficult to convert to CCG combinatory rules, and because of noise introduced by the Treebank conversion process.

### 3.4 CCG Dependency Structures

Dependency structures perform two roles in this article. First, they are used for parser evaluation: The accuracy of a parsing model is measured using precision and recall over CCG predicate–argument dependencies. Second, dependency structures form the core of the **dependency model**: Probabilities are defined over dependency structures, and the parsing algorithm for this model returns the highest scoring dependency structure.

We define a CCG dependency structure as a set of CCG predicate–argument dependencies. They are defined as sets, rather than multisets, because the lexical items in a dependency are considered to be indexed by sentence position; this is important for evaluation purposes and, for the dependency model, determining which derivations lead to a given set of dependencies. However, there are situations where the lexical items need to be considered independently of sentence position, for example when defining feature functions in terms of dependencies. Such cases should be clear from the context.

We define CCG predicate–argument relations in terms of the argument slots in CCG lexical categories. Thus the transitive verb category, $(S\backslash NP)/NP$, has two predicate–argument relations associated with it, one corresponding to the object $NP$ argument and one corresponding to the subject $NP$ argument. In order to distinguish different argument slots, the arguments are numbered from left to right. Thus, the subject relation for a transitive verb is represented as $\langle (S\backslash NP_1)/NP_2, 1\rangle$.

The predicate–argument dependencies are represented as 5-tuples: $\langle h_f, f, s, h_a, l\rangle$, where $h_f$ is the lexical item of the lexical category expressing the dependency relation, $f$ is the lexical category, $s$ is the argument slot, $h_a$ is the head word of the argument, and $l$ encodes whether the dependency is non-local. For example, the dependency encoding *company* as the object of *bought* (as in *IBM bought the company*) is represented as follows:

$$\langle bought_2,\ (S\backslash NP_1)/NP_2,\ 2,\ company_4,\ -\rangle \tag{8}$$

The subscripts on the lexical items indicate sentence position, and the final field ($-$) indicates that the dependency is a local dependency.

Head and dependency information is represented on the lexical categories, and dependencies are created during a derivation as argument slots are filled. Long-range dependencies are created by passing head information from one category to another using unification. For example, the expanded category for the control verb *persuade* is:

$$\text{persuade} := ((S[dcl]_{persuade}\backslash NP_1)/(S[to]_2\backslash NP_X))/NP_{X,3} \tag{9}$$

The head of the infinitival complement's subject is identified with the head of the object, using the variable $X$. Unification then passes the head of the object to the subject of the infinitival, as in standard unification-based accounts of control. In the current implementation, the head and dependency markup depends on the category only and not the lexical item. This gives semantically incorrect dependencies in some cases; for

example, the control verbs *persuade* and *promise* have the same lexical category, which means that *promise Brooks to go* is assigned a structure meaning *promise Brooks that Brooks will go*.

The kinds of lexical items that use the head passing mechanism are raising, auxiliary and control verbs, modifiers, and relative pronouns. Among the constructions that project unbounded dependencies are relativization and right node raising. The following relative pronoun category (for words such as *who, which,* and *that*) shows how heads are co-indexed for object-extraction:

$$\text{who} := (NP_X \backslash NP_{X,1})/(S[dcl]_2/NP_X) \tag{10}$$

In a sentence such as *The company which IBM bought*, the co-indexing will allow *company* to be returned as the object of *bought*, which is represented using the following dependency:

$$\langle bought_2,\ (S\backslash NP_1)/NP_2,\ 2,\ company_4,\ (NP\backslash NP)/(S[dcl]/NP)\rangle \tag{11}$$

The final field indicates the category which mediated the long-range dependency, in this case the object relative pronoun category.

The dependency annotation also permits complex categories as arguments. For example, the marked up category for *about* (as in *about 5,000 pounds*) is:

$$(N_X/N_X)_Y/(N/N)_{Y,1} \tag{12}$$

If *5,000* has the category $(N_X/N_X)_{5,000}$, the dependency relation marked on the $(N/N)_{Y,1}$ argument in (12) allows the dependency between *about* and *5,000* to be captured.

In the current implementation every argument slot in a lexical category corresponds to a dependency relation. This means, for example, that the parser produces subjects of to-infinitival clauses and auxiliary verbs. In the sentence *IBM may like to buy Lotus*, *IBM* will be returned as the subject of *may*, *like*, *to*, and *buy*. The only exception is during evaluation, when some of these dependencies are ignored in order to be consistent with the predicate–argument dependencies in CCGbank, and also DepBank. In future work we may investigate removing some of these dependencies from the parsing model and the parser output.

## 4. Log-Linear Parsing Models for CCG

This section describes two parsing models for CCG. The first defines the probability of a dependency structure, and the second—the normal-form model—defines the probability of a single derivation. In many respects, modeling single derivations is simpler than modeling dependency structures, as the rest of the article will demonstrate. However, there are a number of reasons for modeling dependency structures. First, for many applications predicate–argument dependencies provide a more useful output than derivations, and the parser evaluation is over dependencies; hence it would seem reasonable to optimize over the dependencies rather than the derivation. Second, we want to investigate, for the purposes of parse selection, whether there is useful information in the nonstandard derivations. We can test this by defining the probability of a dependency structure in terms of all the derivations leading to that structure, rather

than emphasising a single derivation. Thus, the probability of a dependency structure, $\pi$, given a sentence, $S$, is defined as follows:

$$P(\pi|S) = \sum_{d \in \Delta(\pi)} P(d, \pi|S) \tag{13}$$

where $\Delta(\pi)$ is the set of derivations which lead to $\pi$.

This approach is different from that of Clark, Hockenmaier, and Steedman (2002), who define the probability of a dependency structure simply in terms of the dependencies. One reason for modeling derivations (either one distinguished derivation or a set of derivations), in addition to predicate–argument dependencies, is that derivations may contain useful information for inferring the correct dependency structure.

For both the dependency model and the normal-form model, the probability of a parse is defined using a log-linear form. However, the meaning of *parse* differs in the two cases. For the dependency model, a parse is taken to be a $\langle d, \pi \rangle$ pair, as in Equation (13). For the normal-form model, a parse is simply a (head-lexicalized) derivation.[2] We define a conditional log-linear model of a parse $\omega \in \Omega$, given a sentence $S$, as follows:

$$P(\omega|S) = \frac{1}{Z_S} e^{\boldsymbol{\lambda} \cdot \boldsymbol{f}(\omega)} \tag{14}$$

where $\boldsymbol{\lambda} \cdot \boldsymbol{f}(\omega) = \sum_i \lambda_i f_i(\omega)$. The function $f_i$ is the integer-valued frequency function of the $i$th feature; $\lambda_i$ is the weight of the $i$th feature; and $Z_S$ is a normalizing constant which ensures that $P(\omega|S)$ is a probability distribution:

$$Z_S = \sum_{\omega' \in \rho(S)} e^{\boldsymbol{\lambda} \cdot \boldsymbol{f}(\omega')} \tag{15}$$

where $\rho(S)$ is the set of possible parses for $S$. For the normal-form model, features are defined over single derivations, including local word–word dependencies arising from lexicalized rule instantiations. The feature set is derived from the gold-standard normal-form derivations in CCGbank. For the dependency model, features are defined over dependency structures as well as derivations, and the feature set is derived from all derivations leading to gold-standard dependency structures, including nonstandard derivations. Section 7 describes the feature types in more detail.

### 4.1 Estimating the Dependency Model

For the dependency model, the training data consists of gold-standard dependency structures, namely, sets of CCG predicate–argument dependencies, as described earlier. We follow Riezler et al. (2002) in using a discriminative estimation method by maximizing the *conditional* log-likelihood of the model given the data, minus a Gaussian prior

2 We could model predicate–argument dependencies together with the derivation, but we wanted to use features from the derivation only, following Hockenmaier and Steedman (2002b).

term to prevent overfitting (Chen and Rosenfeld 1999; Johnson et al. 1999). Thus, given training sentences $S_1, \ldots, S_m$, gold-standard dependency structures, $\pi_1, \ldots, \pi_m$, and the definition of the probability of a dependency structure from Equation (13), the objective function is:

$$\begin{aligned} L'(\Lambda) &= L(\Lambda) - G(\Lambda) \\ &= \log \prod_{j=1}^{m} P_\Lambda(\pi_j|S_j) - \sum_{i=1}^{n} \frac{\lambda_i^2}{2\sigma_i^2} \\ &= \sum_{j=1}^{m} \log \frac{\sum_{d \in \Delta(\pi_j)} e^{\boldsymbol{\lambda} \cdot \boldsymbol{f}(d,\pi_j)}}{\sum_{\omega \in \rho(S_j)} e^{\boldsymbol{\lambda} \cdot \boldsymbol{f}(\omega)}} - \sum_{i=1}^{n} \frac{\lambda_i^2}{2\sigma_i^2} \\ &= \sum_{j=1}^{m} \log \sum_{d \in \Delta(\pi_j)} e^{\boldsymbol{\lambda} \cdot \boldsymbol{f}(d,\pi_j)} - \sum_{j=1}^{m} \log \sum_{\omega \in \rho(S_j)} e^{\boldsymbol{\lambda} \cdot \boldsymbol{f}(\omega)} - \sum_{i=1}^{n} \frac{\lambda_i^2}{2\sigma_i^2} \end{aligned} \tag{16}$$

where $L(\Lambda)$ is the log-likelihood of model $\Lambda$, $G(\Lambda)$ is the Gaussian prior term, and $n$ is the number of features. We use a single smoothing parameter $\sigma$, so that $\sigma_i = \sigma$ for all $i$; however, grouping the features into classes and using a different $\sigma$ for each class is worth investigating and may improve the results.

Optimization of the objective function, whether using iterative scaling or more general numerical optimization methods, requires calculation of the gradient of the objective function at each iteration. The components of the gradient vector are as follows:

$$\begin{aligned} \frac{\partial L'(\Lambda)}{\partial \lambda_i} = &\sum_{j=1}^{m} \sum_{d \in \Delta(\pi_j)} \frac{e^{\boldsymbol{\lambda} \cdot \boldsymbol{f}(d,\pi_j)} f_i(d,\pi_j)}{\sum_{d \in \Delta(\pi_j)} e^{\boldsymbol{\lambda} \cdot \boldsymbol{f}(d,\pi_j)}} \\ &- \sum_{j=1}^{m} \sum_{\omega \in \rho(S_j)} \frac{e^{\boldsymbol{\lambda} \cdot \boldsymbol{f}(\omega)} f_i(\omega)}{\sum_{\omega \in \rho(S_j)} e^{\boldsymbol{\lambda} \cdot \boldsymbol{f}(\omega)}} - \frac{\lambda_i}{\sigma_i^2} \end{aligned} \tag{17}$$

The first two terms are expectations of feature $f_i$: the second expectation is over all derivations for each sentence in the training data, and the first is over only the derivations leading to the gold-standard dependency structure for each sentence.

The estimation process attempts to make the expectations in Equation (17) equal (ignoring the Gaussian prior term). Another way to think of the estimation process is that it attempts to put as much mass as possible on the derivations leading to the gold-standard structures (Riezler et al. 2002). The Gaussian prior term prevents overfitting by penalizing any model whose weights get too large in absolute value.

The estimation process can also be thought of in terms of the framework of Della Pietra, Della Pietra, and Lafferty (1997), because setting the gradient in Equation (17) to zero yields the usual maximum entropy constraints, namely that the expected value of each feature is equal to its empirical value (again ignoring the Gaussian prior term). However, in this case the empirical values are themselves expectations, over all derivations leading to each gold-standard dependency structure.

### 4.2 Estimating the Normal-Form Model

For the normal-form model, the training data consists of gold-standard normal-form derivations. The objective function and gradient vector for the normal-form model are:

$$L'(\Lambda) = L(\Lambda) - G(\Lambda) \tag{18}$$
$$= \log \prod_{j=1}^{m} P_\Lambda(d_j|S_j) - \sum_{i=1}^{n} \frac{\lambda_i^2}{2\sigma_i^2}$$

$$\frac{\partial L'(\Lambda)}{\partial \lambda_i} = \sum_{j=1}^{m} f_i(d_j) \tag{19}$$
$$- \sum_{j=1}^{m} \sum_{d \in \theta(S_j)} \frac{e^{\boldsymbol{\lambda} \cdot \boldsymbol{f}(d)} f_i(d)}{\sum_{d \in \theta(S_j)} e^{\boldsymbol{\lambda} \cdot \boldsymbol{f}(d)}} - \frac{\lambda_i}{\sigma_i^2}$$

where $d_j$ is the the gold-standard normal-form derivation for sentence $S_j$ and $\theta(S_j)$ is the set of possible derivations for $S_j$. Note that $\theta(S_j)$ could contain some non-normal-form derivations; however, because any non-normal-form derivations will be considered incorrect, the resulting model will typically assign low probabilities to non-normal-form derivations.

The empirical value in Equation (19) is simply a count of the number of times the feature appears in the gold-standard normal-form derivations. The second term in Equation (19) is an expectation over all derivations for each sentence.

### 4.3 The Limited-Memory BFGS Algorithm

The limited memory BFGS (L-BFGS) algorithm is a general purpose numerical optimization algorithm (Nocedal and Wright 1999). In contrast to iterative scaling algorithms such as GIS, which update the parameters one at a time on each iteration, L-BFGS updates the parameters all at once on each iteration. It does this by considering the topology of the feature space and moving in a direction which is guaranteed to increase the value of the objective function.

The simplest way in which to consider the shape of the feature space is to move in the direction in which the value of the objective function increases most rapidly; this leads to the method of **steepest-ascent**. Hence steepest-ascent uses the first partial derivative (the gradient) of the objective function to determine parameter updates. L-BFGS improves on steepest-ascent by also considering the second partial derivative (the Hessian). In fact, calculation of the Hessian can be prohibitively expensive, and so L-BFGS estimates this derivative by observing the change in a fixed number of previous gradients (hence the *limited memory*).

Malouf (2002) gives a more thorough description of numerical optimization methods applied to log-linear models. He also presents a convincing demonstration that general purpose numerical optimization methods can greatly outperform iterative scaling methods for many NLP tasks.[3] Malouf uses standard numerical computation libraries

3 One NLP task for which we have found GIS to be especially suitable is sequence tagging, and we still use GIS to estimate tagging models (Curran and Clark 2003).

as the basis of his implementation. One of our aims was to provide a self contained estimation code base, and so we implemented our own version of the L-BFGS algorithm as described in Nocedal and Wright (1999).

## 5. Efficient Estimation

The L-BFGS algorithm requires the following values at each iteration: the expected value and the empirical expected value of each feature, for calculating the gradient; and the value of the likelihood function. For the normal-form model, the empirical expected values and the likelihood can be easily obtained, because these only involve the single gold-standard derivation for each sentence. For the dependency model, the computations of the empirical expected values and the likelihood function are more complex, because these involve sums over just those derivations leading to the gold-standard dependency structures. We explain how these derivations can be found in Section 5.4. The next section explains how CCG charts can be represented in a way which allows efficient estimation.

### 5.1 Packed CCG Charts as Feature Forests

The packed charts perform a number of roles. First, they compactly represent every ⟨derivation, dependency-structure⟩ pair, by grouping together equivalent chart entries. Entries are **equivalent** when they interact in the same manner with both the generation of subsequent parse structure and the statistical parse selection. In practice, this means that equivalent entries have the same span; form the same structures, that is, the remaining derivation plus dependencies, in any subsequent parsing; and generate the same features in any subsequent parsing. Back pointers to the daughters indicate how an individual entry was created, so that any derivation plus dependency structure can be recovered from the chart.

The second role of the packed charts is to allow recovery of the highest scoring derivation or dependency structure without enumerating all derivations. And finally, packed charts are an instance of a **feature forest**, which Miyao and Tsujii (2002) show can be used to efficiently estimate expected values of features, even though the expectation may involve a sum over an exponential number of trees in the forest. One of the contributions of this section is showing how Miyao and Tsujii's feature forest approach can be applied to a particular grammar formalism, namely CCG. As Chiang (2003) points out, Miyao and Tsujii do not provide a way of constructing a feature forest given a sentence, but provide the mathematical tools for estimation once the feature forest has been constructed.

In our packed charts, entries are equivalent when they have the same category type, identical head, and identical unfilled dependencies. The equivalence test must account for heads and *unfilled* dependencies because equivalent entries form the same dependencies in any subsequent parsing. *Individual entries* in the chart are obtained by combining canonical representatives of equivalence classes, using the rules of the grammar. *Equivalence classes* in the chart are sets of equivalent individual entries.

A feature forest $\Phi$ is defined as a tuple $\langle C, D, R, \gamma, \delta \rangle$ where:

- $C$ is a set of conjunctive nodes;
- $D$ is a set of disjunctive nodes;

- $R \subseteq D$ is a set of root disjunctive nodes;
- $\gamma : D \rightarrow 2^C$ is a conjunctive daughter function;
- $\delta : C \rightarrow 2^D$ is a disjunctive daughter function.

The interpretation of a packed chart as a feature forest is straightforward. First, only entries which are part of a derivation spanning the whole sentence are relevant. These entries can be found by traversing the chart top-down, starting with the entries which span the sentence. Individual entries in a cell are the conjunctive nodes, which are either ⟨lexical category, word⟩ pairs at the leaves, or have been obtained by combining two equivalence classes (or applying a unary rule to an equivalence class). The equivalence classes of individual entries are the disjunctive nodes. And finally, the equivalence classes at the roots of the CCG derivations are the root disjunctive nodes.

For each feature function defined over parses (see Section 4) there is a corresponding feature function defined over conjunctive nodes, that is, for each $f_i : \Omega \rightarrow \mathcal{N}$ there is a corresponding $f_i : C \rightarrow \mathcal{N}$ which counts the number of times feature $f_i$ appears on a particular conjunctive node. The value of $f_i$ for a parse is then the sum of the values of $f_i$ for each conjunctive node in the parse.

The features used in the parsing model determine the definition of the equivalence relation used for grouping individual entries. In our models, features are defined in terms of individual dependencies and local rule instantiations, where a rule instantiation is the local tree arising from the application of a rule in the grammar. Note that features can be defined in terms of long-range dependencies, even though such dependencies may involve words which are a long way apart in the sentence. Our earlier definition of equivalence is consistent with these feature types.

As an example, consider the following composition of *will* with *buy* using the forward composition rule:

$$\begin{array}{c} (S[dcl]_{will}\backslash NP)/NP \\ \diagup \qquad \diagdown \\ (S[dcl]_{will}\backslash NP)/(S[b]\backslash NP) \qquad (S[b]_{buy}\backslash NP)/NP \end{array}$$

The equivalence class of the resulting individual entry is determined by the CCG category plus heads, in this case $(S[dcl]_{will}\backslash NP)/NP$, plus the dependencies yet to be filled. The dependencies are not shown, but there are two subject dependencies on the first *NP*, one encoding the subject of *will* and one encoding the subject of *buy*, and there is an object dependency on the second *NP* encoding the object of *buy*. Entries in the same equivalence class are identical for the purposes of creating new dependencies for the remainder of the parsing.

### 5.2 Feature Locality

It is possible to extend the locality of the features beyond single rule instantiations and local dependencies. For example, the definition of equivalence given earlier allows the incorporation of long-range dependencies as features. The equivalence test considers unfilled dependencies which are both local and long-range; thus any individual entries which have different long-range dependencies waiting to be filled will be in different equivalence classes. One of the advantages of log-linear models is that it is easy to include such features; Hockenmaier (2003b) describes the difficulties in including such

features in a generative model. One of the early motivations of the Edinburgh CCG parsing project was to see if the long-range dependencies recovered by a CCG parser could improve the accuracy of a parsing model. In fact, we have found that adding long-range dependencies to any of the models described in this article has no impact on accuracy. One possible explanation is that the long-range dependencies are so rare that a much larger amount of training data would be required for these dependencies to have an impact. Of course the fact that CCG enables recovery of long-range dependencies is still a useful property, even if these dependencies are not currently useful as features, because it improves the utility of the parser output.

There is considerable flexibility in defining the features for a parsing model in our log-linear framework, as the long-range dependency example demonstrates, but the need for dynamic programming for both estimation and decoding reduces the range of features which can be used. Any extension to the "locality" of the features would reduce the effectiveness of the chart packing and any dynamic programming performed over the chart. Two possible extensions, which we have not investigated, include defining dependency features which account for all three elements of the triple in a PP-attachment (Collins and Brooks 1995), and defining a rule feature which includes the grandparent node (Johnson 1998). Another alternative for future work is to compare the dynamic programming approach taken here with the beam-search approach of Collins and Roark (2004), which allows more "global" features.

### 5.3 Calculating Feature Expectations

For estimating both the normal-form model and the dependency model, the following expectation of each feature $f_i$, with respect to some model $\Lambda$, is required:

$$E_\Lambda f_i = \sum_S \frac{1}{Z_S} \sum_{\omega \in \rho(S)} e^{\boldsymbol{\lambda} \cdot \boldsymbol{f}(\omega)} f_i(\omega) \tag{20}$$

where $\rho(S)$ is the set of all parses for sentence $S$, and $\boldsymbol{\lambda}$ is the vector of weights for $\Lambda$.

This is essentially the same calculation for both models, even though for the dependency model, features can be defined in terms of dependencies as well as the derivations. Dependencies can be stored as part of the individual entries (conjunctive nodes) at which they are created; hence all features can be defined in terms of the individual entries which make up the derivations.

Calculating $E_\Lambda f_i$ requires summing over all derivations $\omega$ which include $f_i$ for each sentence $S$ in the training data. The key to performing this sum efficiently is to write the sum in terms of inside and outside scores for each conjunctive node. The inside and outside scores can be defined recursively. If the inside score for a conjunctive node $c$ is denoted $\phi_c$, and the outside score denoted $\psi_c$, then the expected value of $f_i$ can be written as follows:

$$E_\Lambda f_i = \sum_S \frac{1}{Z_S} \sum_{c \in C_S} f_i(c) \phi_c \psi_c \tag{21}$$

where $C_S$ is the set of conjunctive nodes in the packed chart for sentence $S$.

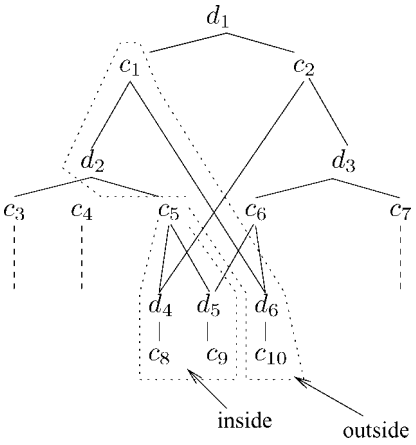



**Figure 4**
Example feature forest.

Figure 4 gives an example feature forest, and shows the nodes used to calculate the inside and outside scores for conjunctive node $c_5$. The inside score for a disjunctive node, $\phi_d$, is the sum of the inside scores for its conjunctive node daughters:

$$\phi_d = \sum_{c \in \gamma(d)} \phi_c \tag{22}$$

The inside score for a conjunctive node, $\phi_c$, is defined in terms of the inside scores of $c$'s disjunctive node daughters:

$$\phi_c = \prod_{d \in \delta(c)} \phi_d \, e^{\boldsymbol{\lambda} \cdot \boldsymbol{f}(c)} \tag{23}$$

where $\boldsymbol{\lambda} \cdot \boldsymbol{f}(c) = \sum_i \lambda_i f_i(c)$. If the conjunctive node is a leaf node, the inside score is just the exponentiation of the sum of the feature weights on that node.

The outside score for a conjunctive node, $\psi_c$, is the outside score for its disjunctive node mother:

$$\psi_c = \psi_d \text{ where } c \in \gamma(d) \tag{24}$$

The calculation of the outside score for a disjunctive node, $\psi_d$, is a little more involved; it is defined as a sum over the conjunctive mother nodes, of the product of the outside score of the mother, the inside score of the disjunctive node sister, and the feature weights on the mother. For example, the outside score of $d_4$ in Figure 4 is the sum of two product terms. The first term is the product of the outside score of $c_5$, the inside score of $d_5$, and the feature weights at $c_5$; and the second term is the product of the outside score of $c_2$, the inside score of $d_3$, and the feature weights at $c_2$. The definition is as follows; the outside score for a root disjunctive node is 1, otherwise:

$$\psi_d = \sum_{\{c | d \in \delta(c)\}} \left( \psi_c \prod_{\{d' | d' \in \delta(c), d' \neq d\}} \phi_{d'} \, e^{\boldsymbol{\lambda} \cdot \boldsymbol{f}(c)} \right) \tag{25}$$

The normalization constant $Z_S$ is the sum of the inside scores for the root disjunctive nodes:

$$Z_S = \sum_{d_r \in R} \phi_{d_r} \tag{26}$$

In order to calculate inside scores, the scores for daughter nodes need to be calculated before the scores for mother nodes (and vice versa for the outside scores). This can easily be achieved by ordering the nodes in the bottom-up CKY parsing order.

### 5.4 Estimation for the Dependency Model

For the dependency model, the computations of the empirical expected values (17) and the log-likelihood function (16) require sums over just those derivations leading to the gold-standard dependency structure. We will refer to such derivations as **correct** derivations. As far as we know, this problem of identifying derivations in a packed chart which lead to a particular dependency structure has not been addressed before in the NLP literature.

Figure 5 gives an algorithm for finding nodes in a packed chart which appear in correct derivations. *cdeps*($c$) returns the number of correct dependencies on conjunctive node $c$, and returns the incorrect marker $*$ if there are any incorrect dependencies on $c$; *dmax*($c$) returns the maximum number of correct dependencies produced by any sub-derivation headed by $c$, and returns $*$ if there are no sub-derivations producing

$\langle C, D, R, \gamma, \delta \rangle$ is a packed chart / feature forest
$G$ is a set of gold-standard dependencies
Let $c$ be a conjunctive node
Let $d$ be a disjunctive node
$deps(c)$ is the set of dependencies on node $c$

$$cdeps(c) = \begin{cases} * \text{ if, for some } \tau \in deps(c), \tau \notin G \\ |deps(c)| \text{ otherwise} \end{cases}$$

$$dmax(c) = \begin{cases} * \text{ if } cdeps(c) == * \\ * \text{ if } dmax(d) == * \text{ for some } d \in \delta(c) \\ \sum_{d \in \delta(c)} dmax(d) + cdeps(c) \text{ otherwise} \end{cases}$$

$$dmax(d) = \max\{dmax(c) \mid c \in \gamma(d)\}$$

**mark**($d$):
  mark $d$ as a correct node
  **foreach** $c \in \gamma(d)$
    **if** $dmax(c) == dmax(d)$
      mark $c$ as a correct node
      **foreach** $d' \in \delta(c)$
        **mark**($d'$)

**foreach** $d_r \in R$ such that $dmax(d_r) = |G|$
  **mark**($d_r$)

**Figure 5**
Algorithm for finding nodes in correct derivations.

only correct dependencies; $dmax(d)$ returns the same value but for disjunctive node $d$. Recursive definitions of these functions are given in Figure 5; the base case occurs when conjunctive nodes have no disjunctive daughters.

The algorithm identifies all those root nodes heading derivations which produce just the correct dependencies, and traverses the chart top-down marking the nodes in those derivations. The insight behind the algorithm is that, for two conjunctive nodes in the same equivalence class, if one node heads a sub-derivation producing more correct dependencies than the other node (and each sub-derivation only produces correct dependencies), then the node with less correct dependencies cannot be part of a correct derivation.

The conjunctive and disjunctive nodes appearing in correct derivations form a new feature forest, which we call a **correct forest**. The correct forest forms a subset of the **complete forest** (containing all derivations for the sentence). The correct and complete forests can be used to estimate the required log-likelihood value and feature expectations. Let $E^{\Phi}_{\Lambda} f_i$ be the expected value of $f_i$ over the forest $\Phi$ for model $\Lambda$; then the values in Equation (17) can be obtained by calculating $E^{\Phi_j}_{\Lambda} f_i$ for the complete forest $\Phi_j$ for each sentence $S_j$ in the training data, and also $E^{\Psi_j}_{\Lambda} f_i$ for each correct forest $\Psi_j$:

$$\frac{\partial L(\Lambda)}{\partial \lambda_i} = \sum_{j=1}^{m} (E^{\Psi_j}_{\Lambda} f_i - E^{\Phi_j}_{\Lambda} f_i) \tag{27}$$

The log-likelihood in Equation (16) can be calculated as follows:

$$L(\Lambda) = \sum_{j=1}^{m} (\log Z_{\Psi_j} - \log Z_{\Phi_j}) \tag{28}$$

where $\log Z_{\Phi}$ and $\log Z_{\Psi}$ are the normalization constants for $\Phi$ and $\Psi$.

### 5.5 Estimation in Practice

Estimating the parsing models consists of generating packed charts for each sentence in the training data, and then repeatedly calculating the values needed by the L-BFGS estimation algorithm until convergence. Even though the packed charts are an efficient representation of the derivation space, the charts for the complete training data (Sections 02–21 of CCGbank) take up a considerable amount of memory. One solution is to only keep a small number of charts in memory at any one time, and to keep reading in the charts on each iteration. However, given that the L-BFGS algorithm takes hundreds of iterations to converge, this approach would be infeasibly slow.

Our solution is to keep all charts in memory by developing a parallel version of the L-BFGS training algorithm and running it on an 18-node Beowulf cluster. As well as solving the memory problem, another significant advantge of parallelization is the reduction in estimation time: using 18 nodes allows our best-performing model to be estimated in less than three hours.

We use the the Message Passing Interface (MPI) standard for the implementation (Gropp et al. 1996). The parallel implementation is a straightforward extension of the BFGS algorithm. Each machine in the cluster deals with a subset of the training data,

holding the packed charts for that subset in memory. The key stages of the algorithm are the calculations of the model expectations and the likelihood function. For a single-process version these are calculated by summing over all the training instances in one place. For a multi-process version, these are summed in parallel, and at the end of each iteration the parallel sums are combined to give a master sum. Producing a master operation across a cluster using MPI is a *reduce* operation. In our case, every node needs to be holding a copy of the master sum, so we use an *all_reduce* operation.

The MPI library handles all aspects of the parallelization, including finding the optimal way of summing across the nodes of the Beowulf cluster (typically it is done using a tree algorithm). In fact, the parallelization only adds around twenty lines of code to the single-process implementation. Because of the simplicity of the parellel communication between the nodes, parallelizing the estimation code is an example of an **embarrassingly parallel** problem. One difficult aspect of the parallel implementation is that debugging can be much harder, in which case it is often easier to test a non-MPI version of the program first.



## 6. The Decoder

For the normal-form model, the Viterbi algorithm is used to find the most probable derivation from a packed chart. For each equivalence class, we record the individual entry at the root of the subderivation which has the highest score for the class. The equivalence classes were defined so that any other individual entry cannot be part of the highest scoring derivation for the sentence. The score for a subderivation $d$ is $\sum_i \lambda_i f_i(d)$ where $f_i(d)$ is the number of times the $i$th feature occurs in the subderivation. The highest-scoring subderivations can be calculated recursively using the highest-scoring equivalence classes that were combined to create the individual entry.

For the dependency model, the highest scoring dependency structure is required. Clark and Curran (2003) outline an algorithm for finding the most probable dependency structure, which keeps track of the highest scoring set of dependencies for each node in the chart. For a set of equivalent entries in the chart (a disjunctive node), this involves summing over all conjunctive node daughters which head sub-derivations leading to the same set of high scoring dependencies. In practice large numbers of such conjunctive nodes lead to very long parse times.

As an alternative to finding the most probable dependency structure, we have developed an algorithm which maximizes the expected labeled recall over dependencies. Our algorithm is based on Goodman's (1996) labeled recall algorithm for the phrase-structure PARSEVAL measures. As far as we know, this is the first application of Goodman's approach to finding highest scoring dependency structures. Watson, Carroll, and Briscoe (2005) have also applied our algorithm to the grammatical relations output by the RASP parser.

The dependency structure, $\pi_{\max}$, which maximizes the expected recall is:

$$\pi_{\max} = \operatorname*{argmax}_{\pi} \sum_{\pi_i} P(\pi_i|S)|\pi \cap \pi_i| \tag{29}$$

where $\pi_i$ ranges over the dependency structures for $S$. The expectation for a single dependency structure $\pi$ is realized as a weighted intersection over all possible dependency structures $\pi_i$ for $S$. The intuition is that, if $\pi_i$ is the gold standard, then the number of dependencies recalled in $\pi$ is $|\pi \cap \pi_i|$. Because we do not know which $\pi_i$ is the gold

standard, then we calculate the *expected* recall by summing the recall of $\pi$ relative to each $\pi_i$, weighted by the probability of $\pi_i$.

The expression can be expanded further:

$$\begin{aligned}\pi_{\max} &= \operatorname*{argmax}_{\pi} \sum_{\pi_i} P(\pi_i|S) \sum_{\tau \in \pi} 1 \text{ if } \tau \in \pi_i \\ &= \operatorname*{argmax}_{\pi} \sum_{\tau \in \pi} \sum_{\pi'|\tau \in \pi'} P(\pi'|S) \\ &= \operatorname*{argmax}_{\pi} \sum_{\tau \in \pi} \sum_{d \in \Delta(\pi')|\tau \in \pi'} P(d|S) \end{aligned} \tag{30}$$

The reason for this manipulation is that the expected recall score for $\pi$ is now written in terms of a sum over the individual dependencies in $\pi$, rather than a sum over each dependency structure for $S$. The inner sum is over all derivations which contain a particular individual dependency $\tau$. Thus the final score for a dependency structure $\pi$ is a sum of the scores for each dependency $\tau$ in $\pi$; and the score for a dependency $\tau$ is the sum of the probabilities of those derivations producing $\tau$. This latter sum can be calculated efficiently using inside and outside scores:

$$\pi_{\max} = \operatorname*{argmax}_{\pi} \sum_{\tau \in \pi} \frac{1}{Z_S} \sum_{c \in C} \phi_c \psi_c \text{ if } \tau \in \mathit{deps}(c) \tag{31}$$

where $\phi_c$ is the inside score and $\psi_c$ is the outside score for node $c$; $C$ is the set of conjunctive nodes in the packed chart for sentence $S$ and $\mathit{deps}(c)$ is the set of dependencies on conjunctive node $c$. The intuition behind the expected recall score is that a dependency structure scores highly if it has dependencies produced by high probability derivations.[4]

The reason for rewriting the score in terms of individual dependencies is to make use of the packed chart: The score for an individual dependency can be calculated using dynamic programming (as explained previously), and the highest scoring dependency structure can be found using dynamic programming also. The algorithm which finds $\pi_{\max}$ is essentially the same as the Viterbi algorithm described earlier, efficiently finding a derivation which produces the highest scoring set of dependencies.

## 7. Model Features

The log-linear modeling framework allows considerable flexibility for representing the parse space in terms of features. In this article we limit the features to those defined over local rule instantiations and single predicate–argument dependencies. The feature sets described below differ for the dependency and normal-form models. The

4 Coordinate constructions can create multiple dependencies for a single argument slot; in this case the score for these multiple dependencies is the average of the individual scores.

**Table 1**
Features common to the dependency and normal-form models.

| Feature type | Example |
|---|---|
| LexCat + Word | $(S/S)/NP$ + Before |
| LexCat + POS | $(S/S)/NP$ + IN |
| RootCat | $S[dcl]$ |
| RootCat + Word | $S[dcl]$ + was |
| RootCat + POS | $S[dcl]$ + VBD |
| Rule | $S[dcl] \rightarrow NP\ S[dcl] \backslash NP$ |
| Rule + Word | $S[dcl] \rightarrow NP\ S[dcl] \backslash NP$ + bought |
| Rule + POS | $S[dcl] \rightarrow NP\ S[dcl] \backslash NP$ + VBD |

dependency model has features defined over the CCG predicate–argument dependencies, whereas the dependencies for the normal-form model are defined in terms of local rule instantiations in the derivation. Another difference is that the rule features for the normal-form model are taken from the gold-standard normal-form derivations, whereas the dependency model contains rule features from non-normal-form derivations.

There are a number of features defined over derivations which are common to the dependency model and the normal-form model.[5] First, there are features which represent each ⟨word, lexical-category⟩ pair in a derivation, and generalizations of these which represent ⟨POS, lexical-category⟩ pairs. Second, there are features representing the root category of a derivation, which we also extend with the head word of the root category; this latter feature is then generalized using the POS tag of the head (as previously described). Third, there are features which encode rule instantiations—local trees consisting of a parent and one or two children—in the derivation. The first set of rule features encode the combining categories and the result category; the second set of features extend the first by also encoding the head of the result category; and the third set generalizes the second using POS tags. Table 1 gives an example for each of these feature types.

The dependency model also has CCG predicate–argument dependencies as features, defined as 5-tuples as in Section 3.4. In addition these features are generalized in three ways using POS tags, with the word–word pair replaced with word–POS, POS–word, and POS–POS. Table 2 gives some examples.

We extend the dependency features further by adding distance information. The distance features encode the dependency relation and the word associated with the lexical category (but not the argument word), plus some measure of distance between the two dependent words. We use three distance measures which count the following: the number of intervening words, with four possible values 0, 1, 2, or more; the number of intervening punctuation marks, with four possible values 0, 1, 2, or more; and the number of intervening verbs (determined by POS tag), with three possible values 0, 1, or more. Each of these features is again generalized by replacing the word associated with the lexical category with its POS tag.

5 Each feature has a corresponding frequency function, defined in Equation (14), which counts the number of times the feature appears in a derivation.

**Table 2**
Predicate–argument dependency features for the dependency model.

| Feature type | Example |
|---|---|
| Word–Word | $\langle bought, (S\backslash NP_1)/NP_2, 2, stake, (NP\backslash NP)/(S[dcl]/NP)\rangle$ |
| Word–POS | $\langle bought, (S\backslash NP_1)/NP_2, 2, \text{NN}, (NP\backslash NP)/(S[dcl]/NP)\rangle$ |
| POS–Word | $\langle \text{VBD}, (S\backslash NP_1)/NP_2, 2, stake, (NP\backslash NP)/(S[dcl]/NP)\rangle$ |
| POS–POS | $\langle \text{VBD}, (S\backslash NP_1)/NP_2, 2, \text{NN}, (NP\backslash NP)/(S[dcl]/NP)\rangle$ |
| Word + Distance(words) | $\langle bought, (S\backslash NP_1)/NP_2, 2, (NP\backslash NP)/(S[dcl]/NP)\rangle + 2$ |
| Word + Distance(punct) | $\langle bought, (S\backslash NP_1)/NP_2, 2, (NP\backslash NP)/(S[dcl]/NP)\rangle + 0$ |
| Word + Distance(verbs) | $\langle bought, (S\backslash NP_1)/NP_2, 2, (NP\backslash NP)/(S[dcl]/NP)\rangle + 0$ |
| POS + Distance(words) | $\langle \text{VBD}, (S\backslash NP_1)/NP_2, 2, (NP\backslash NP)/(S[dcl]/NP)\rangle + 2$ |
| POS + Distance(punct) | $\langle \text{VBD}, (S\backslash NP_1)/NP_2, 2, (NP\backslash NP)/(S[dcl]/NP)\rangle + 0$ |
| POS + Distance(verbs) | $\langle \text{VBD}, (S\backslash NP_1)/NP_2, 2, (NP\backslash NP)/(S[dcl]/NP)\rangle + 0$ |

**Table 3**
Rule dependency features for the normal-form model.

| Feature type | Example |
|---|---|
| Word–Word | $\langle company, S[dcl]\rightarrow NP\ S[dcl]\backslash NP, bought\rangle$ |
| Word–POS | $\langle company, S[dcl]\rightarrow NP\ S[dcl]\backslash NP, \text{VBD}\rangle$ |
| POS–Word | $\langle \text{NN}, S[dcl]\rightarrow NP\ S[dcl]\backslash NP, bought\rangle$ |
| POS–POS | $\langle \text{NN}, S[dcl]\rightarrow NP\ S[dcl]\backslash NP, \text{VBD}\rangle$ |
| Word + Distance(words) | $\langle bought, S[dcl]\rightarrow NP\ S[dcl]\backslash NP\rangle + > 2$ |
| Word + Distance(punct) | $\langle bought, S[dcl]\rightarrow NP\ S[dcl]\backslash NP\rangle + 2$ |
| Word + Distance(verbs) | $\langle bought, S[dcl]\rightarrow NP\ S[dcl]\backslash NP\rangle + 0$ |
| POS + Distance(words) | $\langle \text{VBD}, S[dcl]\rightarrow NP\ S[dcl]\backslash NP\rangle + > 2$ |
| POS + Distance(punct) | $\langle \text{VBD}, S[dcl]\rightarrow NP\ S[dcl]\backslash NP\rangle + 2$ |
| POS + Distance(verbs) | $\langle \text{VBD}, S[dcl]\rightarrow NP\ S[dcl]\backslash NP\rangle + 0$ |

For the normal-form model we follow Hockenmaier and Steedman (2002b) by defining dependency features in terms of the local rule instantiations, by adding the heads of the combining categories to the rule instantiation features.[6] These are generalized in three ways using POS tags, as shown in Table 3. There are also the three distance measures which encode the distance between the two head words of the combining categories, as for the dependency model. Here the distance feature encodes the combining categories, the result category, the head of the result category (either as a word or POS tag), and the distance between the two head words.

For the features in the normal-form model, a frequency cutoff of two was applied; that is, a feature had to occur at least twice in the gold-standard normal-form derivations to be included in the model. The same cutoff was applied to the features in the dependency model, except for the rule instantiation feature types. For these features the counting was done across all derivations licensed by the gold-standard lexical category sequences and a frequency cutoff of 10 was applied. The larger cutoff was used because the productivity of the grammar can lead to very large numbers of these features. We

6 We have also considered a model containing predicate–argument dependencies as well as local rule dependencies, but adding the extra dependency feature types had no impact on the accuracy of the normal-form model.

also only included those features which had a nonzero empirical count, that is, those features which occured on at least one correct derivation. These feature types and frequency cutoffs led to 475,537 features for the normal-form model and 632,591 features for the dependency model.

## 8. The Supertagger

Parsing with lexicalized grammar formalisms such as CCG is a two-step process: first, elementary syntactic structures—in CCG's case lexical categories—are assigned to each word in the sentence, and then the parser combines the structures together. The first step can be performed by simply assigning to each word all lexical categories the word is seen with in the training data, together with some strategy for dealing with rare and unknown words (such as assigning the complete lexical category set; Hockenmaier 2003a). Because the number of lexical categories assigned to a word can be high, some strategy is needed to make parsing practical; Hockenmaier, for example, uses a beam search to discard chart entries with low scores.

In this article we take a different approach, by using a supertagger (Bangalore and Joshi 1999) to perform step one. Clark and Curran (2004a) describe the supertagger, which uses log-linear models to define a distribution over the lexical category set for each local five-word context containing the target word (Ratnaparkhi 1996). The features used in the models are the words and POS tags in the five-word window, plus the two previously assigned lexical categories to the left. The conditional probability of a sequence of lexical categories, given a sentence, is then defined as the product of the individual probabilities for each category.

The most probable lexical category sequence can be found efficiently using a variant of the Viterbi algorithm for HMM taggers. We restrict the categories which can be assigned to a word by using a **tag dictionary:** for words seen at least $k$ times in the training data, the tagger can only assign categories which have been seen with the word in the data. For words seen less than $k$ times, an alternative based on the word's POS tag is used: The tagger can only assign categories which have been seen with the POS tag in the data. We have found the tag dictionary to be beneficial in terms of both efficiency and accuracy. A value of $k = 20$ was used in the experiments described in this article.

The lexical category set used by the supertagger is described in Clark and Curran (2004a) and Curran, Clark, and Vadas (2006). It includes all lexical catgeories which appear at least 10 times in Sections 02–21 of CCGbank, resulting in a set of 425 categories. The Clark and Curran paper shows this set to have very high coverage on unseen data.

The accuracy of the supertagger on Section 00 of CCGbank is 92.6%, with a sentence accuracy of 36.8%. Sentence accuracy is the percentage of sentences whose words are all tagged correctly. These figures include punctuation marks, for which the lexical category is simply the punctuation mark itself, and are obtained using gold standard POS tags. With automatically assigned POS tags, using the POS tagger of Curran and Clark (2003), the accuracies drop to 91.5% and 32.5%. An accuracy of 91–92% may appear reasonable given the large lexical category set; however, the low sentence accuracy suggests that the supertagger may not be accurate enough to serve as a front-end to a parser. Clark (2002) reports that a significant loss in coverage results if the supertagger is used as a front-end to the parser of Hockenmaier and Steedman (2002b). In order to increase the number of words assigned the correct category, we develop a CCG multitagger, which is able to assign more than one category to each word.

**Table 4**
Supertagger ambiguity and accuracy on Section 00.

| β | $k$ | CATS/WORD | ACC | SENT ACC | ACC (POS) | SENT ACC |
|---|---|---|---|---|---|---|
| 0.075 | 20 | 1.27 | 97.34 | 67.43 | 96.34 | 60.27 |
| 0.030 | 20 | 1.43 | 97.92 | 72.87 | 97.05 | 65.50 |
| 0.010 | 20 | 1.72 | 98.37 | 77.73 | 97.63 | 70.52 |
| 0.005 | 20 | 1.98 | 98.52 | 79.25 | 97.86 | 72.24 |
| 0.001 | 150 | 3.57 | 99.17 | 87.19 | 98.66 | 80.24 |

The multitagger uses the following conditional probabilities:

$$P(y_i|w_1,\ldots,w_n) = \sum_{y_1,\ldots,y_{i-1},y_{i+1},\ldots,y_n} P(y_i,y_1,\ldots,y_{i-1},y_{i+1},\ldots,y_n|w_1,\ldots,w_n) \quad (32)$$

Here $y_i$ is to be thought of as a constant category, whereas $y_j$ ($j \neq i$) varies over the possible categories for word $j$. In words, the probability of category $y_i$, given the sentence, is the sum of the probabilities of all sequences containing $y_i$. This sum can be calculated efficiently using a variant of the forward–backward algorithm. For each word in the sentence, the multitagger then assigns all those categories whose probability according to Equation (32) is within some factor, β, of the highest probability category for that word. In the implementation used here the forward–backward sum is limited to those sequences allowed by the tag dictionary. For efficiency purposes, an extra pruning strategy is also used to discard low probability sub-sequences before the forward–backward algorithm is run. This uses a second variable-width beam of $0.1\beta$.

Table 4 gives the per-word accuracy of the supertagger on Section 00 for various levels of category ambiguity, together with the average number of categories per word.[7] The SENT column gives the percentage of sentences whose words are all supertagged correctly. The set of categories assigned to a word is considered correct if it contains the correct category. The table gives results when using gold standard POS tags and, in the final two columns, when using POS tags automatically assigned by the POS tagger described in Curran and Clark (2003). The drop in accuracy is expected, given the importance of POS tags as features.

The table demonstrates the significant reduction in the average number of categories that can be achieved through the use of a supertagger. To give one example, the number of categories in the tag dictionary's entry for the word *is* is 45. However, in the sentence *Mr. Vinken is chairman of Elsevier N.V., the Dutch publishing group.*, the supertagger correctly assigns one category to *is* for all values of β.

In our earlier work (Clark and Curran 2004a) the forward–backward algorithm was not used to estimate the probability in Equation (32). Curran, Clark, and Vadas (2006) investigate the improvement obtained from using the forward–backward algorithm, and also address the drop in supertagger accuracy when using automatically assigned POS tags. We show how to maintain some POS ambiguity through to the supertagging phase, using a multi-POS tagger, and also how POS tag probabilities can be encoded as real-valued features in the supertagger. The drop in supertagging accuracy when

7 The β values used here are slightly different to the values used in earlier publications because the pruning strategy used in the supertagger has changed slightly.

moving from gold to automatically assigned POS tags is reduced by roughly 50% across the various values of β.

## 9. Parsing in Practice

### 9.1 Combining the Supertagger and the Parser

The philosophy in earlier work which combined the supertagger and parser (Clark, Hockenmaier, and Steedman 2002; Clark and Curran 2003) was to use an unrestrictive setting of the supertagger, but still allow a reasonable compromise between speed and accuracy. The idea was to give the parser the greatest possibility of finding the correct parse, by initializing it with as many lexical categories as possible, but still retain reasonable efficiency. However, for some sentences, the number of categories in the chart gets extremely large with this approach, and parsing is unacceptably slow. Hence a limit was applied to the number of categories in the chart, and a more restrictive setting of the supertagger was reverted to if the limit was exceeded.

In this article we consider the opposite approach: Start with a very restrictive setting of the supertagger, and only assign more categories if the parser cannot find an analysis spanning the sentence. In this way the parser interacts much more closely with the supertagger. In effect, the parser is using the grammar to decide if the categories provided by the supertagger are acceptable, and if not the parser requests more categories. The advantage of this *adaptive* supertagging approach is that parsing speeds are much higher, without any corresponding loss in accuracy. Section 10.3 gives results for the speed of the parser.

### 9.2 Chart Parsing Algorithm

The algorithm used to build the packed charts is the CKY chart parsing algorithm (Kasami 1965; Younger 1967) described in Steedman (2000). The CKY algorithm applies naturally to CCG because the grammar is binary. It builds the chart bottom-up, starting with constituents spanning a single word, incrementally increasing the span until the whole sentence is covered. Because the constituents are built in order of span size, at any point in the process all the sub-constituents which could be used to create a particular new constituent must be present in the chart. Hence dynamic programming can be used to prevent the need for backtracking during the parsing process.

### 9.3 Grammar Implementation

There is a trade-off between the size and coverage of the grammar and the efficiency of the parser. One of our main goals in this work has been to develop a parser which can provide analyses for the vast majority of linguistic constructions in CCGbank, but is also efficient enough for large-scale NLP applications. In this section we describe some of the decisions we made when implementing the grammar, with this trade-off in mind.

First, the lexical category set we use does not contain all the categories in Sections 02–21 of CCGbank. Applying a frequency cutoff of 10 results in a set of 425 lexical categories. This set has excellent coverage on unseen data (Clark and Curran 2004a) and is a manageable size for adding the head and dependency information, and also mapping to grammatical relations for evaluation purposes (Section 11).

Second, for the normal-form model, and also the hybrid dependency model described in Section 10.2.1, two types of contraints on the grammar rules are used. Section 3 described the Eisner constraints, in which any constituent which is the result of a forward composition cannot serve as the primary (left) functor in another forward composition or forward application; an analogous constraint applies for backward composition. The second type of constraint only allows two categories to combine if they have been seen to combine in the training data. Although this constraint only permits category combinations seen in Sections 02–21 of CCGbank, we have found that it is detrimental to neither parser accuracy nor coverage.

Neither of these constraints guarantee a normal-form derivation, but they are both effective at reducing the size of the charts, which can greatly increase parser speed (Clark and Curran 2004a). The constraints are also useful for training. Section 10 shows that having a less restrictive setting on the supertagger, when creating charts for discriminative training, can lead to more accurate models. However, the optimal setting on the supertagger for training purposes can only be used when the constraints are applied, because otherwise the memory requirements are prohibitive.

Following Steedman (2000), we place the following constraint on backward crossed composition (for all models): The $Y$ category in (7) cannot be an $N$ or $NP$ category. We also place a similar constraint on backward composition. Both constraints reduce the size of the charts considerably with no impact on coverage or accuracy.

Type-raising is performed by the parser for the categories $NP$, $PP$, and $S[adj]\backslash NP$. It is implemented by adding one of three fixed sets of categories to the chart whenever an $NP$, $PP$, or $S[adj]\backslash NP$ is present. Appendix A gives the category sets. Each category transformation is an instance of the following two rule schemata:

$$X \Rightarrow_{\mathbf{T}} T/(T\backslash X) \quad (> \mathbf{T}) \tag{33}$$

$$X \Rightarrow_{\mathbf{T}} T\backslash(T/X) \quad (< \mathbf{T}) \tag{34}$$

Appendix A lists the punctuation and type-changing rules implemented in the parser. This is a larger grammar than we have used in previous articles (Clark and Curran 2004b, 2004a, 2006), mainly because the improvement in the supertagger since the earlier work means that we can now use a larger grammar but still maintain highly efficient parsing.

## 10. Experiments

The statistics relating to model estimation were obtained using Sections 02–21 of CCGbank as training data. The results for parsing accuracy were obtained using Section 00 as development data and Section 23 as the final test data. The results for parsing speed were obtained using Section 23. There are various hyperparameters in the parsing system, for example the frequency cutoff for features, the $\sigma$ parameter in the Gaussian prior term, the $\beta$ values used in the supertagger, and so on. All of these were set experimentally using Section 00 as development data.

### 10.1 Model Estimation

The gold standard for the normal-form model consists of the normal-form derivations in CCGbank. For the dependency model, the gold-standard dependency structures are

produced by running our CCG parser over the normal-form derivations. It is essential that the packed charts for each sentence contain the gold standard; for the normal-form model this means that our parser must be able to produce the gold-standard derivation from the gold-standard lexical category sequence; and for the dependency model this means that at least one derivation in the chart must produce the gold-standard dependency structure. Not all rule instantiations in CCGbank can be produced by our parser, because some are not instances of combinatory rules, and others are very rare punctuation and type-changing rules which we have not implemented. Hence it is not possible for the parser to produce the gold standard for every sentence in Sections 02–21, for either the normal-form or the dependency model. These sentences are not used in the training process.

For parsing the training data, we ensure that the correct category is a member of the set assigned to each word. (We do not do this when parsing the test data.) The average number of categories assigned to each word is determined by the β parameter in the supertagger. A category is assigned to a word if the category's probability is within β of the highest probability category for that word. Hence the value of β has a direct effect on the size of the packed charts: Smaller β values lead to larger charts.

For training purposes, the β parameter determines how many incorrect derivations will be used for each sentence for the discriminative training algorithm. We have found that the β parameter can have a large impact on the accuracy of the resulting models: If the β value is too large, then the training algorithm does not have enough incorrect derivations to "discriminate against"; if the β value is too small, then this introduces too many incorrect derivations into the training process, and can lead to impractical memory requirements.

For some sentences, the packed charts can become very large. The supertagging approach we adopt for training differs from that used for testing and follows the original approach of Clark, Hockenmaier, and Steedman (2002): If the size of the chart exceeds some threshold, the value of β is increased, reducing ambiguity, and the sentence is supertagged and parsed again. The threshold which limits the size of the charts was set at 300,000 individual entries. (This is the threshold used for training; a higher value was used for testing.) For a small number of long sentences the threshold is exceeded even at the largest β value; these sentences are not used for training.

For the normal-form model we were able to use 35,732 sentences for training (90.2% of Sections 02–21) and for the dependency model 35,889 sentences (90.6%). Table 5 gives training statistics for the normal-form and dependency models (and a hybrid model described in Section 10.2.1), for various sequences of β values, when the training algorithm is run to convergence on an 18-node cluster. The training algorithm is defined to have converged when the percentage change in the objective function is less than 0.0001%. The σ value in Equation (16), which was determined experimentally using the development data, was set at 1.3 for all the experiments in this article.

**Table 5**
Training statistics.

| Model | β values | CPU time (min.) | Iterations | RAM (GB) |
|---|---|---|---|---|
| Dependency | 0.1 | 176.0 | 750 | 24.4 |
| Normal-form | 0.1 | 17.2 | 420 | 5.3 |
| Normal-form | 0.0045, 0.0055, 0.01, 0.05, 0.1 | 72.1 | 466 | 16.1 |
| Hybrid | 0.0045, 0.0055, 0.01, 0.05, 0.1 | 128.4 | 610 | 22.5 |

The main reason that the normal-form model requires less memory and converges faster than the dependency model is that, for the normal-form model, we applied the two types of normal-form restriction described in Section 9.3: First, categories can only combine if they appear together in a rule instantiation in Sections 2–21 of CCGbank; and second, we applied the Eisner constraints described in Section 3.

We conclude this section by noting that it is only through the use of the supertagger that we are able to perform the discriminative estimation at all; without it the memory requirements would be prohibitive, even when using the cluster.

### 10.2 Parsing Accuracy

This section gives accuracy figures on the predicate–argument dependencies in CCGbank. Overall results are given, as well as results broken down by relation type, as in Clark, Hockenmaier, and Steedman (2002). Because the purpose of this article is to demonstrate the feasibility of wide-coverage parsing with CCG, we do not give an evaluation targeted specifically at long-range dependencies; such an evaluation was presented in Clark, Steedman, and Curran (2004).

For evaluation purposes, the threshold parameter which limits the size of the charts was set at 1,000,000 individual entries. This value was chosen to maximize the coverage of the parser, so that the evaluation is performed on as much of the unseen data as possible. This was also the threshold parameter used for the speed experiments in Section 10.3.

All of the intermediate results were obtained using Section 00 of CCGbank as development data. The final test result, showing the performance of the best performing model, was obtained using Section 23. Evaluation was performed by comparing the dependency output of the parser against the predicate–argument dependencies in CCGbank. We report precision, recall, and F-scores for labeled and unlabeled dependencies, and also category accuracy. The category accuracy is the percentage of words assigned the correct lexical category by the parser (including punctuation). The labeled dependency scores take into account the lexical category containing the dependency relation, the argument slot, the word associated with the lexical category, and the argument head word: All four must be correct to score a point. For the unlabeled scores, only the two dependent words are considered. The F-score is the balanced harmonic mean of precision ($P$) and recall ($R$): $2PR/(P + R)$. The scores are given only for those sentences which were parsed successfully. We also give coverage values showing the percentage of sentences which were parsed successfully.

Using the CCGbank dependencies for evaluation is a departure from our earlier work, in which we generated our own gold standard by running the parser over the derivations in CCGbank and outputting the dependencies. In this article we wanted to use a gold standard which is easily accessible to other researchers. However, there are some differences between the dependency scheme used by our parser and CCGbank. For example, our parser outputs some coordination dependencies which are not in CCGbank; also, because the parser currently encodes every argument slot in each lexical category as a dependency relation, there are some relations, such as the subject of *to* in a *to*-infinitival construction, which are not in CCGbank either. In order to provide a fair evaluation, we ignore those dependency relations. This still leaves some minor differences. We can measure the remaining differences as follows: Comparing the CCGbank dependencies in Section 00 against those generated by running our parser over the derivations in 00 gives labeled precision and recall values of 99.80% and 99.18%,

respectively. Thus there are a small number of dependencies in CCGbank which the current version of the parser can never get right.

*10.2.1 Dependency Model vs. Normal-Form Model.* Table 6 shows the results for the normal-form and dependency models evaluated against the predicate–argument dependencies in CCGbank. Gold standard POS tags were used; the LF(POS) column gives the labeled F-score with automatically assigned POS tags for comparison. Decoding with the dependency model involves finding the maximum-recall dependency structure, and decoding with the normal-form model involves finding the most probable derivation, as described in Section 6. The β value refers to the setting of the supertagger used for training and is the first in the sequence of βs from Table 5. The β values used during the testing are those in Table 4 and the new, efficient supertagging strategy of taking the highest β value first was used.

With the same β values used for training ($\beta = 0.1$), the results for the dependency model are slightly higher than for the normal-form model. However, the coverage of the normal-form model is higher (because the use of the normal-form constraints mean that there are less sentences which exceed the chart-size threshold). One clear result from the table is that increasing the chart size used for training, by using smaller β values, can significantly improve the results, in this case around 1.5% F-score for the normal-form model.

The training of the dependency model already uses most of the RAM available on the cluster. However, it is possible to use smaller β values for training the dependency model if we also apply the two types of normal-form restriction used by the normal-form model. This hybrid model still uses the features from the dependency model; it is still trained using dependency structures as the gold standard; and decoding is still performed using the maximum-recall algorithm; the only difference is that the derivations in the charts are restricted by the normal-form constraints (both for training and testing). Table 5 gives the training statistics for this model, compared to the dependency and normal-form models. The number of sentences we were able to use for training this model was 36,345 (91.8% of Sections 02–21). The accuracy of this hybrid dependency model is given in Table 7. These are the highest results we have obtained to date on Section 00. We also give the results for the normal-form model from Table 6 for comparison.

Table 8 gives the results for the hybrid dependency model, broken down by relation type, using the same relations given in Clark, Hockenmaier, and Steedman (2002). Automatically assigned POS tags were used.

*10.2.2 Final Test Results.* Table 9 gives the final test results on Section 23 for the hybrid dependency model. The coverage for these results is 99.63% (for gold-standard POS

**Table 6**
Results for dependency and normal-form models on Section 00.

| Model | β | LP | LR | LF | LF (POS) | SENT ACC | UP | UR | UF | CAT ACC | cov |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Dependency | 0.1 | 86.52 | 84.97 | 85.73 | 84.24 | 32.06 | 92.91 | 91.24 | 92.07 | 93.37 | 98.17 |
| Normal-form | 0.1 | 85.50 | 84.68 | 85.08 | 83.34 | 31.93 | 92.38 | 91.49 | 91.93 | 93.04 | 99.06 |
| Normal-form | 0.0045 | 87.17 | 86.30 | 86.73 | 84.74 | 34.99 | 93.21 | 92.28 | 92.74 | 94.05 | 99.06 |

**Table 7**
Results on Section 00 with both models using normal-form constraints.

| Model | β | LP | LR | LF | LF (POS) | SENT ACC | UP | UR | UF | CAT ACC | cov |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Normal-form | 0.0045 | 87.17 | 86.30 | 86.73 | 84.74 | 34.99 | 93.21 | 92.28 | 92.74 | 94.05 | 99.06 |
| Hybrid dependency | 0.0045 | 88.06 | 86.43 | 87.24 | 85.25 | 35.67 | 93.88 | 92.13 | 93.00 | 94.16 | 99.06 |

**Table 8**
Results for the hybrid dependency model on Section 00 by dependency relation.

| Lexical category | Arg Slot | | LP % | # deps | LR % | # deps | F-score |
|---|---|---|---|---|---|---|---|
| $N_X/N_{X,1}$ | 1 | *nominal modifier* | 95.28 | 7,314 | 95.62 | 7,288 | 95.45 |
| $NP_X/N_{X,1}$ | 1 | *determiner* | 96.57 | 4,078 | 96.03 | 4,101 | 96.30 |
| $(NP_X\backslash NP_{X,1})/NP_2$ | 2 | *np modifying prep* | 82.17 | 2,574 | 88.90 | 2,379 | 85.40 |
| $(NP_X\backslash NP_{X,1})/NP_2$ | 1 | *np modifying prep* | 81.58 | 2,285 | 85.74 | 2,174 | 83.61 |
| $((S_X\backslash NP_Y)\backslash(S_{X,1}\backslash NP_Y))/NP_2$ | 2 | *vp modifying prep* | 71.94 | 1,169 | 73.32 | 1,147 | 72.63 |
| $((S_X\backslash NP_Y)\backslash(S_{X,1}\backslash NP_Y))/NP_2$ | 1 | *vp modifying prep* | 70.92 | 1,073 | 71.93 | 1,058 | 71.42 |
| $(S[dcl]\backslash NP_1)/NP_2$ | 1 | *transitive verb* | 81.62 | 914 | 85.55 | 872 | 83.54 |
| $(S[dcl]\backslash NP_1)/NP_2$ | 2 | *transitive verb* | 81.57 | 971 | 86.37 | 917 | 83.90 |
| $(S_X\backslash NP_Y)\backslash(S_{X,1}\backslash NP_Y)$ | 1 | *adverbial modifier* | 86.85 | 745 | 86.73 | 746 | 86.79 |
| $PP/NP_1$ | 1 | *prep complement* | 75.06 | 818 | 70.09 | 876 | 72.49 |
| $(S[b]\backslash NP_1)/NP_2$ | 2 | *inf transitive verb* | 84.01 | 663 | 87.03 | 640 | 85.50 |
| $(S[dcl]\backslash NP_{X,1})/(S[b]_2\backslash NP_X)$ | 2 | *auxiliary* | 97.70 | 478 | 97.90 | 477 | 97.80 |
| $(S[dcl]\backslash NP_{X,1})/(S[b]_2\backslash NP_X)$ | 1 | *auxiliary* | 94.15 | 479 | 92.99 | 485 | 93.57 |
| $(S[b]\backslash NP_1)/NP_2$ | 1 | *inf transitive verb* | 77.82 | 496 | 73.95 | 522 | 75.83 |
| $(NP_X/N_{X,1})\backslash NP_2$ | 1 | *s genitive* | 96.57 | 379 | 95.56 | 383 | 96.06 |
| $(NP_X/N_{X,1})\backslash NP_2$ | 2 | *s genitive* | 97.35 | 377 | 98.66 | 372 | 98.00 |
| $(S[dcl]\backslash NP_1)/S[dcl]_2$ | 1 | *sentential comp verb* | 94.88 | 371 | 90.96 | 387 | 92.88 |
| $(NP_X\backslash NP_{X,1})/(S[dcl]_2\backslash NP_X)$ | 1 | *subject rel pronoun* | 85.77 | 260 | 81.39 | 274 | 83.52 |
| $(NP_X\backslash NP_{X,1})/(S[dcl]_2\backslash NP_X)$ | 2 | *subject rel pronoun* | 97.45 | 275 | 97.10 | 276 | 97.28 |
| $(NP_X\backslash NP_{X,1})/(S[dcl]_2/NP_X)$ | 1 | *object rel pronoun* | 81.82 | 22 | 69.23 | 26 | 75.00 |
| $(NP_X\backslash NP_{X,1})/(S[dcl]_2/NP_X)$ | 2 | *object rel pronoun* | 86.36 | 22 | 82.61 | 23 | 84.44 |
| $NP/(S[dcl]_1/NP)$ | 1 | *headless obj rel pron* | 100.00 | 17 | 100.00 | 17 | 100.00 |

**Table 9**
Results for the hybrid dependency model on Section 23.

| | LP | LR | LF | SENT | UP | UR | UF | CAT ACC | cov |
|---|---|---|---|---|---|---|---|---|---|
| Hybrid dependency | 88.34 | 86.96 | 87.64 | 36.53 | 93.74 | 92.28 | 93.00 | 94.32 | 99.63 |
| Hybrid dependency (POS) | 86.17 | 84.74 | 85.45 | 32.92 | 92.43 | 90.89 | 91.65 | 92.98 | 99.58 |
| Hockenmaier (2003) | 84.3 | 84.6 | 84.4 | – | 91.8 | 92.2 | 92.0 | 92.2 | 99.83 |
| Hockenmaier (POS) | 83.1 | 83.5 | 83.3 | – | 91.1 | 91.5 | 91.3 | 91.5 | 99.83 |

tags), which corresponds to 2,398 of the 2,407 sentences in Section 23 receiving an analysis. When using automatically assigned POS tags, the coverage is slightly lower: 99.58%. We used version 1.2 of CCGbank to obtain these results. Results are also given for Hockenmaier's parser (Hockenmaier 2003a) which used an earlier, slightly different version of the treebank. We wanted to use the latest version to enable other researchers to compare with our results.

### 10.3 Parse Times

The results in this section were obtained using a 3.2 GHz Intel Xeon P4. Table 10 gives parse times for the 2,407 sentences in Section 23 of CCGbank. In order not to optimize speed by compromising accuracy, we used the hybrid dependency model, together with both kinds of normal-form constraints, and the maximum-recall decoder. Times are given for both automatically assigned POS tags and gold-standard POS tags (POS). The sents and words columns give the number of sentences, and the number of words, parsed per second. For all of the figures reported on Section 23, unless stated otherwise, we chose settings for the various parameters which resulted in a coverage of 99.6%. It is possible to obtain an analysis for the remaining 0.4%, but at a significant loss in speed. The parse times and speeds include the failed sentences, and include the time taken by the supertagger, but not the POS tagger; however, the POS tagger is extremely efficient, taking less than 4 seconds to supertag Section 23, most of which consists of load time for the Maximum Entropy model.

The first row corresponds to the strategy of earlier work by starting with an unrestrictive setting of the supertagger. The first value of $\beta$ is 0.005; if the parser cannot find a spanning analysis, this is changed to $\beta = 0.001_{k=150}$, which increases the average number of categories assigned to a word by decreasing $\beta$ and increasing the tag-dictionary parameter. If the node limit is exceeded at $\beta = 0.005$ (for these experiments the node limit is set at 1,000,000), $\beta$ is changed to 0.01. If the node limit is still exceeded, $\beta$ is changed to 0.03, and finally to 0.075.

The second row corresponds to the new strategy of starting with the most restrictive setting of the supertagger ($\beta = 0.075$), and moving through the settings if the parser cannot find a spanning analysis. The table shows that the new strategy has a significant impact on parsing speed, increasing it by a factor of 3 over the earlier approach (given the parameter settings used in these experiments).

The penultimate row corresponds to using only one supertagging level with $\beta = 0.075$; the parser ignores the sentence if it cannot get an analysis at this level. The percentage of sentences without an analysis is now over 6% (with automatically assigned POS tags), but the parser is extremely fast, processing over 30 sentences per second. This configuration of the system would be useful for obtaining data for lexical knowledge acquisition, for example, for which large amounts of data are required. The oracle row gives the parser speed when it is provided with only the correct lexical categories, showing the speeds which could be achieved given the perfect supertagger.

Table 11 gives the percentage of sentences which are parsed at each supertagger level, for both the new and old parsing strategies. The results show that, for the old approach, most of the sentences are parsed using the least restrictive setting of the supertagger ($\beta = 0.005$); conversely, for the new approach, most of the sentences are

**Table 10**
Parse times for Section 23.

| Supertagging/parsing constraints | Time (sec) | Sents/ sec | Words/ sec | Time (POS) (sec) | Sents (POS)/ sec | Words (POS)/ sec |
|---|---|---|---|---|---|---|
| $\beta = 0.005 \rightarrow \ldots \rightarrow 0.075$ | 379.4 | 6.3 | 145.9 | 369.0 | 6.5 | 150.0 |
| $\beta = 0.075 \rightarrow \ldots 0.001_{k=150}$ | 99.9 | 24.1 | 554.3 | 116.6 | 20.6 | 474.7 |
| $\beta = 0.075$ (95.3/93.4% cov) | 79.7 | 30.2 | 694.6 | 79.8 | 30.1 | 693.5 |
| Oracle (94.7% cov) | 23.8 | 101.0 | 2,322.6 | | | |

**Table 11**
Supertagger β levels used on Section 00.

| | | 0.075 FIRST | | 0.005 FIRST | |
|---|---|---|---|---|---|
| β | CATS/WORD | PARSES | % | PARSES | % |
| 0.075 | 1.27 | 1,786 | 93.4 | 3 | 0.2 |
| 0.03 | 1.43 | 45 | 2.4 | 5 | 0.3 |
| 0.01 | 1.72 | 24 | 1.3 | 2 | 0.1 |
| 0.005 | 1.98 | 18 | 0.9 | 1,863 | 97.4 |
| $0.001_{k=150}$ | 3.57 | 22 | 1.2 | 22 | 1.2 |
| FAIL | | 18 | 0.9 | 18 | 0.9 |

**Table 12**
Comparing parser speeds on Section 23 of the WSJ Penn Treebank.

| Parser | Time (min.) |
|---|---|
| Collins | 45 |
| Charniak | 28 |
| Sagae | 11 |
| CCG | 1.9 |

parsed using the most restrictive setting ($\beta = 0.075$). This suggests that, in order to increase the accuracy of the parser without losing efficiency, the accuracy of the supertagger at the $\beta = 0.075$ level needs to be improved, without increasing the number of categories assigned on average.

A possible response to our policy of adaptive supertagging is that any statistical parser can be made to run faster, for example by changing the beam parameter in the Collins (2003) parser, but that any increase in speed is typically associated with a reduction in accuracy. For the CCG parser, the accuracy did not degrade when using the new adaptive parsing strategy. Thus the accuracy and efficiency of the parser were not tuned separately: The configuration used to obtain the speed results was also used to obtain the accuracy results in Sections 10.2 and 11.

To give some idea of how these parsing speeds compare with existing parsers, Table 12 gives the parse times on Section 23 for a number of well-known parsers. Sagae and Lavie (2005) is a classifier-based linear time parser. The times for the Sagae, Collins, and Charniak parsers were taken from the Sagae and Lavie paper, and were obtained using a 1.8 GHz P4, compared to a 3.2 GHz P4 for the CCG numbers. Comparing parser speeds is especially problematic because of implementation differences and the fact that the accuracy of the parsers is not being controlled. Thus we are not making any strong claims about the efficiency of parsing with CCG compared to other formalisms. However, the results in Table 12 add considerable weight to one of our main claims in this article, namely, that highly efficient parsing is possible with CCG, and that large-scale processing is possible with linguistically motivated grammars.

## 11. Cross-Formalism Comparison

An obvious question is how well the CCG parser compares with parsers using different grammar formalisms. One question we are often asked is whether the CCG derivations

output by the parser could be converted to Penn Treebank–style trees to enable a comparison with, for example, the Collins and Charniak parsers. The difficulty is that CCG derivations often have a different shape to the Penn Treebank analyses (coordination being a prime example) and reversing the mapping used by Hockenmaier to create CCGbank is a far from trivial task.

There is some existing work comparing parser performance across formalisms. Briscoe and Carroll (2006) evaluate the RASP parser on the Parc Dependency Bank (DepBank; King et al. 2003). Cahill et al. (2004) evaluate an LFG parser, which uses an automatically extracted grammar, against DepBank. Miyao and Tsujii (2004) evaluate their HPSG parser against PropBank (Palmer, Gildea, and Kingsbury 2005). Kaplan et al. (2004) compare the Collins parser with the Parc LFG parser by mapping Penn Treebank parses into the dependencies of DepBank, claiming that the LFG parser is more accurate with only a slight reduction in speed. Preiss (2003) compares the parsers of Collins and Charniak, the grammatical relations finder of Buchholz, Veenstra, and Daelemans (1999), and the Briscoe and Carroll (2002) parser, using the gold-standard grammatical relations (GRs) from Carroll, Briscoe, and Sanfilippo (1998). The Penn Treebank trees of the Collins and Charniak parsers, and the GRs of the Buchholz parser, are mapped into the required grammatical relations, with the result that the GR finder of Buchholz is the most accurate.

There are a number of problems with such evaluations. The first is that, when converting the output of the Collins parser, for example, into the output of another parser, the Collins parser is at an immediate disadvantage. This is especially true if the alternative output is significantly different from the Penn Treebank trees and if the information required to produce the alternative output is hard to extract. One could argue that the relative lack of grammatical information in the output of the Collins parser is a weakness and any evaluation should measure that. However, we feel that the onus of mapping into another formalism should ideally lie with the researchers making claims about their own particular parser. The second difficulty is that some constructions may be analyzed differently across formalisms, and even apparently trivial differences such as tokenization can complicate the comparison (Crouch et al. 2002).

Despite these difficulties we have attempted a cross-formalism comparison of the CCG parser. For the gold standard we chose the version of DepBank reannotated by Briscoe and Carroll (2006) (hereafter B&C), consisting of 700 sentences from Section 23 of the Penn Treebank. The B&C scheme is similar to the original DepBank scheme in many respects, but overall contains less grammatical detail. Briscoe and Carroll (2006) describe the differences between the two schemes.

We chose this resource for the following reasons: It is publicly available, allowing other researchers to compare against our results; the GRs making up the annotation share some similarities with the predicate–argument dependencies output by the CCG parser; and we can directly compare our parser against a non-CCG parser, namely the RASP parser—and because we are converting the CCG output into the format used by RASP the CCG parser is not at an unfair advantage. There is also the SUSANNE GR gold standard (Carroll, Briscoe, and Sanfilippo 1998), on which the B&C annotation is based, but we chose not to use this for evaluation. This earlier GR scheme is less like the dependencies output by the CCG parser, and the comparison would be complicated further by fact that, unlike CCGbank, the SUSANNE corpus is not based on the Penn Treebank.

The GRs are described in Briscoe (2006), Briscoe and Carroll (2006), and Briscoe, Carroll, and Watson (2006). Table 13 contains the complete list of GRs used in the evaluation, with examples taken from Briscoe. The CCG dependencies were transformed into GRs in two stages. The first stage was to create a mapping between the

**Table 13**
The grammatical relations scheme used in the cross-formalism comparison.

| GR | Description | Example | Relevant GRs in example |
|---|---|---|---|
| conj | coordinator | *Kim likes oranges, apples, and satsumas or clementines* | (dobj likes and)<br>(conj and oranges)<br>(conj and apples)<br>(conj and or)<br>(conj or satsumas)<br>(conj or clementines) |
| aux | auxiliary | *Kim has been sleeping* | (aux sleeping has)<br>(aux sleeping been) |
| det | determiner | *the man* | (det man the) |
| ncmod | non-clausal modifier | *the old man in the barn slept* | (ncmod _ man old)<br>(ncmod _ man in)<br>(dobj in barn) |
| | | *the butcher's shop* | (ncmod poss shop butcher) |
| xmod | unsaturated predicative modifier | *who to talk to* | (xmod to who talk)<br>(iobj talk to)<br>(dobj to who) |
| cmod | saturated clausal modifier | *although he came, Kim left* | (cmod _ left although)<br>(ccomp although came) |
| ncsubj | non-clausal subject | *Kim left* | (ncsubj left Kim _) |
| | | *the upset man* | (ncsubj upset man obj)<br>(passive upset) |
| | | *Kim wants to go* | (ncsubj go Kim _) |
| | | *He's going said Kim* | (ncsubj said Kim inv) |
| passive | passive verb | *issues were filed* | (passive filed)<br>(ncsubj filed issues obj) |
| xsubj | unsaturated predicative subject | *leaving matters* | (xsubj matters leaving _) |
| csubj | saturated clausal subject | *that he came matters* | (csubj matters came that) |
| dobj | direct object | *she gave it to Kim* | (dobj gave it)<br>(iobj gave to)<br>(dobj to Kim) |
| obj2 | second object | *she gave Kim toys* | (obj2 gave toys)<br>(dobj gave Kim) |
| iobj | indirect object | *Kim flew to Paris from Geneva* | (iobj flew to)<br>(iobj flew from)<br>(dobj to Paris)<br>(dobj from Geneva) |
| pcomp | PP which is a PP complement | *Kim climbed through into the attic* | (pcomp climbed through)<br>(pcomp through into)<br>(dobj into attic) |
| xcomp | unsaturated VP complement | *Kim thought of leaving* | (xcomp _ thought of)<br>(xcomp _ of leaving) |
| ccomp | saturated clausal complement | *Kim asked about him playing rugby* | (ccomp _ asked about)<br>(ccomp _ about him)<br>(ncsubj playing him _)<br>(dobj playing rugby) |
| ta | textual adjunct delimited by punctuation | *He made the discovery: Kim was the abbot* | (ta colon discovery was) |

CCG dependencies and the GRs. This involved mapping each argument slot in the 425 lexical categories in the CCG lexicon onto a GR. In the second stage, the GRs created for a particular sentence—by applying the mapping to the parser output—were passed through a Python script designed to correct some of the obvious remaining differences between the CCG and GR representations.

In the process of performing the transformation we encountered a methodological problem: Without looking at examples it was difficult to create the mapping and impossible to know whether the two representations were converging. Briscoe, Carroll,

and Watson (2006) split the 700 sentences in DepBank into a test and development set, but the latter only consists of 140 sentences which we found was not enough to reliably create the transformation. There are some development files in the RASP release which provide examples of the GRs, which we used when possible, but these only cover a subset of the CCG lexical categories.

Our solution to this problem was to convert the gold-standard dependencies from CCGbank into GRs and use these to develop the transformation. So we did inspect the annotation in DepBank, and compared it to the transformed CCG dependencies, but only the gold-standard CCG dependencies. Thus the parser output was never used during this process. We also ensured that the dependency mapping and the post-processing are general to the GRs scheme and not specific to the test set. Table 14 gives some examples of the dependency mapping. Because the number of sentences annotated with GRs is so small, the only other option would have been to guess at various DepBank analyses, which would have made the the evaluation even more biased against the CCG parser.

One advantage of this approach is that, by comparing the transformed gold-standard CCG dependencies with the gold-standard GRs, we can measure how close the CCG representation is to the GRs. This provides some indication of how difficult it is to perform the transformation, and also provides an upper bound on the accuracy of the parser on DepBank. This method would be useful when converting the output of the Collins parser into an alternative representation (Kaplan et al. 2004): Applying the transformation to the gold-standard Penn Treebank trees and comparing with DepBank would provide an upper bound on the performance of the Collins parser and give some indication of the effectiveness of the transformation.

### 11.1 Mapping the CCG Dependencies to GRs

Table 14 gives some examples of the mapping. In our notation, %l indicates the word associated with the lexical category and %f is the head of the constituent filling the

**Table 14**
Examples of the CCG dependency to GRs mapping; $l denotes the word associated with the lexical category and $f is the filler.

| CCG lexical category | arg slot | GR |
|---|---|---|
| $(S[dcl]\backslash NP_1)/NP_2$ | 1 | `(ncsubj %l %f _)` |
| $(S[dcl]\backslash NP_1)/NP_2$ | 2 | `(dobj %l %f)` |
| $(S\backslash NP)/(S\backslash NP)_1$ | 1 | `(ncmod _ %f %l)` |
| $(NP\backslash NP_1)/NP_2$ | 1 | `(ncmod _ %f %l)` |
| $(NP\backslash NP_1)/NP_2$ | 2 | `(dobj %l %f)` |
| $NP[nb]/N_1$ | 1 | `(det %f %l)` |
| $(NP\backslash NP_1)/(S[pss]\backslash NP)_2$ | 1 | `(xmod _ %f %l)` |
| $(NP\backslash NP_1)/(S[pss]\backslash NP)_2$ | 2 | `(xcomp _ %l %f)` |
| $((S\backslash NP)\backslash(S\backslash NP)_1)/S[dcl]_2$ | 1 | `(cmod _ %f %l)` |
| $((S\backslash NP)\backslash(S\backslash NP)_1)/S[dcl]_2$ | 2 | `(ccomp _ %l %f)` |
| $(S[dcl]\backslash NP_1)/(S[adj]\backslash NP)_2$ | 1 | `(ncsubj %l %f _)` |
| $(S[dcl]\backslash NP_1)/(S[adj]\backslash NP)_2$ | 2 | `(xcomp _ %l %f)` |
| $((S[dcl]\backslash NP_1)/NP_2)/NP_3$ | 1 | `(ncsubj %l %f _)` |
| $((S[dcl]\backslash NP_1)/NP_2)/NP_3$ | 2 | `(obj2 %l %f)` |
| $((S[dcl]\backslash NP_1)/NP_2)/NP_3$ | 3 | `(dobj %l %f)` |
| $(S[dcl]\backslash NP_1)/(S[b]\backslash NP)_2$ | 2 | `(aux %f %l)` |

argument slot. For many of the CCG dependencies, the mapping into GRs is straightforward. For example, the first two rows of Table 14 show the mapping for the transitive verb category $(S[dcl]\backslash NP_1)/NP_2$: Argument slot 1 is a non-clausal subject and argument slot 2 is a direct object. In the example *Kim likes juicy oranges*, *likes* is associated with the transitive verb category, *Kim* is the subject, and *oranges* is the head of the constituent filling the object slot, leading to the following GRs: `(ncsubj likes Kim _)` and `(dobj likes oranges)`. The third row shows an example of a modifier: $(S\backslash NP)/(S\backslash NP)$ modifies a verb phrase to the right. Note that, in this example, the order of the lexical category (%l) and filler (%f) is switched compared to the previous example to match the DepBank annotation.

There are a number of reasons why creating the dependency transformation is more difficult than these examples suggest. The first problem is that the mapping from CCG dependencies to GRs is many-to-many. For example, the transitive verb category $(S[dcl]\backslash NP)/NP$ applies to the copular in sentences like *Imperial Corp. is the parent of Imperial Savings & Loan*. With the default annotation the relation between *is* and *parent* would be `dobj`, whereas in DepBank the argument of the copular is analyzed as an `xcomp`. Table 15 gives some examples of how we attempt to deal with this problem. The constraint in the first example means that, whenever the word associated with the transitive verb category is a form of *be*, the second argument is `xcomp`, otherwise the default case applies (in this case `dobj`). There are a number of categories with similar constraints, checking whether the word associated with the category is a form of *be*.

The second type of constraint, shown in the third line of the table, checks the lexical category of the word filling the argument slot. In this example, if the lexical category of the preposition is $PP/NP$, then the second argument of $(S[dcl]\backslash NP)/PP$ maps to `iobj`; thus in *The loss stems from several factors* the relation between the verb and preposition is `(iobj stems from)`. If the lexical category of the preposition is $PP/(S[ng]\backslash NP)$, then the GR is `xcomp`; thus in *The future depends on building cooperation* the relation between the verb and preposition is `(xcomp _ depends on)`. There are a number of CCG dependencies with similar constraints, many of them covering the `iobj/xcomp` distinction.

The second difficulty in creating the transformation is that not all the GRs are binary relations, whereas the CCG dependencies are all binary. The primary example of this is to-infinitival constructions. For example, in the sentence *The company wants to wean itself away from expensive gimmicks*, the CCG parser produces two dependencies relating *wants*, *to* and *wean*, whereas there is only one GR: `(xcomp to wants wean)`. The final row of

**Table 15**
Examples of the many-to-many nature of the CCG dependency to GRs mapping, and a terniary GR.

| CCG lexical category | Slot | GR | Constraint | Example |
|---|---|---|---|---|
| $(S[dcl]\backslash NP_1)/NP_2$ | 2 | `(xcomp _ %l %f)` | word=be | *The parent* **is Imperial** |
| | | `(dobj %l %f)` | | *The parent* **sold Imperial** |
| $(S[dcl]\backslash NP_1)/PP_2$ | 2 | `(iobj %l %f)` | cat=$PP/NP$ | *The loss* **stems from** *several factors* |
| | | `(xcomp _ %l %f)` | cat=$PP/(S[ng]\backslash NP)$ | *The future* **depends on** *building cooperation* |
| $(S[dcl]\backslash NP_1)/(S[to]\backslash NP)_2$ | 2 | `(xcomp %f %l %k)` | cat=$(S[to]\backslash NP)/(S[b]\backslash NP)$ | **wants to wean itself** *away from* |

Table 15 gives an example. We implement this constraint by introducing a %k variable into the GR template which denotes the argument of the category in the constraint column (which, as before, is the lexical category of the word filling the argument slot). In the example, the current category is $(S[dcl]\backslash NP_1)/(S[to]\backslash NP)_2$, which is associated with *wants*; this combines with $(S[to]\backslash NP)/(S[b]\backslash NP)$, associated with *to*; and the argument of $(S[to]\backslash NP)/(S[b]\backslash NP)$ is *wean*. The %k variable allows us to look beyond the arguments of the current category when creating the GRs.

A further difficulty in creating the transformation is that the head passing conventions differ between DepBank and CCGbank. By "head passing" we mean the mechanism which determines the heads of constituents and the mechanism by which words become arguments of long-range dependencies. For example, in the sentence *The group said it would consider withholding royalty payments*, the DepBank and CCGbank annotations create a dependency between *said* and the following clause. However, in DepBank the relation is between *said* and *consider*, whereas in CCGbank the relation is between *said* and *would*. We fixed this problem by changing the head of *would consider* to be *consider* rather than *would*. In practice this means changing the annotation of all the relevant lexical categories in the *markedup* file.[8] The majority of the categories to which this applies are those creating aux relations.

A related difference between the two resources is that there are more subject relations in CCGbank than DepBank. In the previous example, CCGbank has a subject relation between *it* and *consider*, and also *it* and *would*, whereas DepBank only has the relation between *it* and *consider*. In practice this means ignoring a number of the subject dependencies output by the CCG parser, which is implemented by annotating the relevant lexical categories plus argument slot in the *markedup* file with an "ignore" marker.

Another example where the dependencies differ in the two resources is the treatment of relative pronouns. For example, in *Sen. Mitchell, who had proposed the streamlining*, the subject of *proposed* is *Mitchell* in CCGbank but *who* in DepBank. Again, we implemented this change by fixing the head annotation in the lexical categories which apply to relative pronouns.

In summary, considerable changes were required to the *markedup* file in order to bring the dependency annotations of CCGbank and DepBank closer together. The major types of changes have been described here, but not all the details.

### 11.2 Post-Processing of the GR Output

Despite the considerable changes made to the parser output described in the previous section, there were still significant differences between the GRs created from the CCG dependencies and the DepBank GRs. To obtain some idea of whether the schemes were converging, we performed the following oracle experiment. We took the CCG derivations from CCGbank corresponding to the sentences in DepBank, and ran the parser over the gold-standard derivations, outputting the newly created GRs.[9] Treating the DepBank GRs as a gold standard, and comparing these with the CCGbank GRs,

8 The *markedup* file is the file containing the lexical categories, together with annotation which determines dependency and head information, plus the CCG dependency to GR mapping. Appendix B shows part of the *markedup* file.

9 All GRs involving a punctuation mark were removed because the RASP evaluation script can only handle tokens which appear in the gold-standard GRs for the sentence.

gave precision and recall scores of only 76.23% and 79.56%, respectively. Thus given the current mapping, the perfect CCGbank parser would achieve an F-score of only 77.86% when evaluated against DepBank.

On inspecting the output, it was clear that a number of general rules could be applied to bring the schemes closer together, which we implemented as a Python post-processing script. We now provide a description of some of the major changes, to give an indication of the kinds of rules we implemented. We tried to keep the changes as general as possible and not specific to the test set, although some rules, such as the handling of monetary amounts, are genre-specific. We decided to include these rules because they are trivial to implement and significantly affect the score, and we felt that, without these changes, the CCG parser would be unfairly penalized.

The first set of changes deals with coordination. One significant difference between DepBank and CCGbank is the treatment of coordinations as arguments. Consider the example *The president and chief executive officer said the loss stems from several factors.* In both CCGbank and DepBank there are two `conj` GRs arising from the coordination: `(conj and president)` and `(conj and officer)`.[10] The difference arises in the subject of *said*: in DepBank the subject is *and*: `(ncsubj said and _)`, whereas in CCGbank there are two subjects: `(ncsubj said president _)` and `(ncsubj said officer _)`. We deal with this problem by replacing any pairs of GRs which differ only in their arguments, and where the arguments are coordinated items, with a single GR containing the coordination term as the argument. Two arguments are coordinated if they appear in `conj` relations with the same coordinating term, where "same term" is determined by both the word and sentence position.

Another source of `conj` errors is coordination terms acting as sentential modifiers, with category $S/S$, often at the beginning of a sentence. These are labeled `conj` in DepBank, but the GR for $S/S$ is `ncmod`. So any `ncmod` whose modifier's lexical category is $S/S$, and whose POS tag is CC, is changed to `conj`.

Ampersands are also a significant problem, and occur frequently in WSJ text. For example, the CCGbank analysis of *Standard & Poor's index* assigns the lexical category $N/N$ to both *Standard* and *&*, treating them as modifiers of *Poor*, whereas DepBank treats *&* as a coordinating term. We fixed this by creating `conj` GRs between any *&* and the two words on either side; removing the modifier GR between the two words; and replacing any GRs in which the words on either side of the *&* are arguments with a single GR in which *&* is the argument.

The `ta` relation, which identifies text adjuncts delimited by punctuation (Briscoe 2006), is difficult to assign correctly to the parser output. The simple punctuation rules used by the parser, and derived from CCGbank, do not contain enough information to distinguish between the various cases of `ta`. Thus the only rule we have implemented, which is somewhat specific to the newspaper genre, is to replace GRs of the form `(cmod _ say arg)` with `(ta quote arg say)`, where `say` can be any of *say*, *said*, or *says*. This rule applies to only a small subset of the `ta` cases but has high enough precision to be worthy of inclusion.

A common source of error is the distinction between `iobj` and `ncmod`, which is not surprising given the difficulty that human annotators have in distinguishing arguments and adjuncts. There are many cases where an argument in DepBank is an adjunct in CCGbank, and vice versa. The only change we have made is to turn all `ncmod` GRs with

10 CCGbank does not contain GRs in this form, although we will continue to talk as though it does; these are the GRs after the CCGbank dependencies have been put through the dependency to GRs mapping.

*of* as the modifier into iobj GRs (unless the ncmod is a partitive predeterminer). This was found to have high precision and applies to a significant number of cases.

There are some dependencies in CCGbank which do not appear in DepBank. Examples include any dependencies in which a punctuation mark is one of the arguments, and so we removed these from the output of the parser.

We have made some attempt to fill the **subtype slot** for some GRs. The subtype slot specifies additional information about the GR; examples include the value obj in a passive ncsubj, indicating that the subject is an underlying object; the value num in ncmod, indicating a numerical quantity; and prt in ncmod to indicate a verb particle. The passive case is identified as follows: Any lexical category which starts $S[pss]\backslash NP$ indicates a passive verb, and we also mark any verbs POS tagged VBN and assigned the lexical category $N/N$ as passive. Both these rules have high precision, but still leave many of the cases in DepBank unidentified. Many of those remaining are POS tagged JJ and assigned the lexical category $N/N$, but this is also true of many non-passive modifiers, so we did not attempt to extend these rules further. The numerical case is identified using two rules: the num subtype is added if any argument in a GR is assigned the lexical category $N/N[num]$, and if any of the arguments in an ncmod is POS tagged CD. prt is added to an ncmod if the modifiee has a POS tag beginning V and if the modifier has POS tag RP.

We are not advocating that any of these post-processing rules should form part of a parser. It would be preferable to have the required information in the treebank from which the grammar is extracted, so that it could be integrated into the parser in a principled way. However, in order that the parser evaluation be as fair and informative as possible, it is important that the parser output conform as closely to the gold standard as possible. Thus it is appropriate to use any general transformation rules, as long as they are simple and not specific to the test set, to achieve this.

The final columns of Table 16 show the accuracy of the transformed gold-standard CCGbank dependencies when compared with DepBank; the simple post-processing rules have increased the F-score from 77.86% to 84.76%. However, note that this F-score provides an *upper bound* on the performance of the CCG parser, and that this score is still below the F-scores reported earlier when evaluating the parser output against CCGbank. Section 11.4 contains more discussion of this issue.

### 11.3 Results

The results in Table 16 were obtained by parsing the sentences from CCGbank corresponding to those in the 560-sentence test set used by Briscoe, Carroll, and Watson (2006). We used the CCGbank sentences because these differ in some ways from the original Penn Treebank sentences (there are no quotation marks in CCGbank, for example) and the parser has been trained on CCGbank. Even here we experienced some unexpected difficulties, because some of the tokenization is different between DepBank and CCGbank (even though both resources are based on the Penn Treebank), and there are some sentences in DepBank which have been significantly shortened (for no apparent reason) compared to the original Penn Treebank sentences. We modified the CCGbank sentences—and the CCGbank analyses because these were used for the oracle experiments—to be as close to the DepBank sentences as possible. All the results were obtained using the RASP evaluation scripts, with the results for the RASP parser taken from Briscoe, Carroll, and Watson (2006). The results for CCGbank were obtained using the oracle method described previously.

**Table 16**
Accuracy on DepBank.

| | RASP | | | CCG parser | | | CCGbank | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Relation | Prec | Rec | F | Prec | Rec | F | Prec | Rec | F | # GRs |
| dependent | 79.76 | 77.49 | 78.61 | 84.07 | 82.19 | 83.12 | 88.83 | 84.19 | 86.44 | 10,696 |
| aux | 93.33 | 91.00 | 92.15 | 95.03 | 90.75 | 92.84 | 96.47 | 90.33 | 93.30 | 400 |
| conj | 72.39 | 72.27 | 72.33 | 79.02 | 75.97 | 77.46 | 83.07 | 80.27 | 81.65 | 595 |
| ta | 42.61 | 51.37 | 46.58 | 51.52 | 11.64 | 18.99 | 62.07 | 12.59 | 20.93 | 292 |
| det | 87.73 | 90.48 | 89.09 | 95.23 | 94.97 | 95.10 | 97.27 | 94.09 | 95.66 | 1,114 |
| arg_mod | 79.18 | 75.47 | 77.28 | 81.46 | 81.76 | 81.61 | 86.75 | 84.19 | 85.45 | 8,295 |
| mod | 74.43 | 67.78 | 70.95 | 71.30 | 77.23 | 74.14 | 77.83 | 79.65 | 78.73 | 3,908 |
| ncmod | 75.72 | 69.94 | 72.72 | 73.36 | 78.96 | 76.05 | 78.88 | 80.64 | 79.75 | 3,550 |
| xmod | 53.21 | 46.63 | 49.70 | 42.67 | 53.93 | 47.64 | 56.54 | 60.67 | 58.54 | 178 |
| cmod | 45.95 | 30.36 | 36.56 | 51.34 | 57.14 | 54.08 | 64.77 | 69.09 | 66.86 | 168 |
| pmod | 30.77 | 33.33 | 32.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 12 |
| arg | 77.42 | 76.45 | 76.94 | 85.76 | 80.01 | 82.78 | 89.79 | 82.91 | 86.21 | 4,387 |
| subj_or_dobj | 82.36 | 74.51 | 78.24 | 86.08 | 83.08 | 84.56 | 91.01 | 85.29 | 88.06 | 3,127 |
| subj | 78.55 | 66.91 | 72.27 | 84.08 | 75.57 | 79.60 | 89.07 | 78.43 | 83.41 | 1,363 |
| ncsubj | 79.16 | 67.06 | 72.61 | 83.89 | 75.78 | 79.63 | 88.86 | 78.51 | 83.37 | 1,354 |
| xsubj | 33.33 | 28.57 | 30.77 | 0.00 | 0.00 | 0.00 | 50.00 | 28.57 | 36.36 | 7 |
| csubj | 12.50 | 50.00 | 20.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 2 |
| comp | 75.89 | 79.53 | 77.67 | 86.16 | 81.71 | 83.88 | 89.92 | 84.74 | 87.25 | 3,024 |
| obj | 79.49 | 79.42 | 79.46 | 86.30 | 83.08 | 84.66 | 90.42 | 85.52 | 87.90 | 2,328 |
| dobj | 83.63 | 79.08 | 81.29 | 87.01 | 88.44 | 87.71 | 92.11 | 90.32 | 91.21 | 1,764 |
| obj2 | 23.08 | 30.00 | 26.09 | 68.42 | 65.00 | 66.67 | 66.67 | 60.00 | 63.16 | 20 |
| iobj | 70.77 | 76.10 | 73.34 | 83.22 | 65.63 | 73.38 | 83.59 | 69.81 | 76.08 | 544 |
| clausal | 60.98 | 74.40 | 67.02 | 77.67 | 72.47 | 74.98 | 80.35 | 77.54 | 78.92 | 672 |
| xcomp | 76.88 | 77.69 | 77.28 | 77.69 | 74.02 | 75.81 | 80.00 | 78.49 | 79.24 | 381 |
| ccomp | 46.44 | 69.42 | 55.55 | 77.27 | 70.10 | 73.51 | 80.81 | 76.31 | 78.49 | 291 |
| pcomp | 72.73 | 66.67 | 69.57 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 24 |
| macroaverage | 62.12 | 63.77 | 62.94 | 65.71 | 62.29 | 63.95 | 71.73 | 65.85 | 68.67 | |
| microaverage | 77.66 | 74.98 | 76.29 | 81.95 | 80.35 | 81.14 | 86.86 | 82.75 | 84.76 | |

The CCG parser results are based on automatically assigned POS tags, using the Curran and Clark (2003) tagger. For the parser we used the hybrid dependency model and the maximum recall decoder, because this obtained the highest accuracy on CCGbank, with the same parser and supertagger parameter settings as described in Section 10.2.[11] The coverage of the parser on DepBank is 100%. The coverage of the RASP parser is also 100%: 84% of the analyses are complete parses rooted in *S* and the rest are obtained using a robustness technique based on fragmentary analyses (Briscoe and Carroll 2006). The coverage for the oracle experiments is less than 100% (around 95%) since there are some gold-standard derivations in CCGbank which the parser is unable to follow exactly, because the grammar rules used by the parser are a subset of those in CCGbank. The oracle figures are based only on those sentences for which there is a gold-standard analysis, because we wanted to measure how close the two resources are and provide an approximate upper bound for the parser. (But, to repeat, the accuracy figures for the parser are based on the complete test set.)

11 The results reported in Clark and Curran (2007) differ from those here because Clark and Curran used the normal-form model and Viterbi decoder.

F-score is the balanced harmonic mean of precision ($P$) and recall ($R$): $2PR/(P+R)$. # GRs is the number of GRs in DepBank. For a GR in the parser output to be correct, it has to match the gold-standard GR exactly, including any subtype slots; however, it is possible for a GR to be incorrect at one level but correct at a subsuming level. For example, if an `ncmod` GR is incorrectly labeled with `xmod`, but is otherwise correct, it will be correct for all levels which subsume both `ncmod` and `xmod`, for example `mod`. Thus the scores at the most general level in the GR hierarchy (`dependent`) correspond to unlabeled accuracy scores. The micro-averaged scores are calculated by aggregating the counts for all the relations in the hierarchy, whereas the macro-averaged scores are the mean of the individual scores for each relation (Briscoe, Carroll, and Watson 2006).

The results show that the performance of the CCG parser is higher than RASP overall, and also higher on the majority of GR types. Relations on which the CCG parser performs particularly well, relative to RASP, are `conj`, `det`, `ncmod`, `cmod`, `ncsubj`, `dobj`, `obj2`, and `ccomp`. The relations for which the CCG parser performs poorly are some of the less frequent relations: `ta`, `pmod`, `xsubj`, `csubj`, and `pcomp`; in fact `pmod` and `pcomp` are not in the current CCG dependencies to GRs mapping. The overall F-score for the CCG parser, 81.14%, is only 3.6 points below that for CCGbank, which provides an upper bound for the CCG parser.

Briscoe and Carroll (2006) give a rough comparison of RASP with the Parc LFG parser (Kaplan et al. 2004) on DepBank, obtaining similar results overall, but acknowledging that the results are not strictly comparable because of the different annotation schemes used.

### 11.4 Discussion

We might expect the CCG parser to perform better than RASP on this data because RASP is not tuned to newspaper text and uses an unlexicalized parsing model. On the other hand the relatively low upper bound for the CCG parser on DepBank demonstrates the considerable disadvantage of evaluating on a resource which uses a different annotation scheme to the parser. Our feeling is that the overall F-score on DepBank understates the accuracy of the CCG parser, because of the information lost in the translation.

One aspect of the CCGbank evaluation which is more demanding than the DepBank evaluation is the set of labeled dependencies used. In CCGbank there are many more labeled dependencies than GRs in DepBank, because a dependency is defined as a lexical category-argument slot pair. In CCGbank there is a distinction between the direct object of a transitive verb and ditransitive verb, for example, whereas in DepBank these would both be `dobj`. In other words, to get a dependency correct in the CCGbank evaluation, the lexical category—typically a subcategorization frame—has to be correct. In a final experiment we used the GRs generated by transforming CCGbank as a gold standard, against which we compared the GRs from the transformed parser output. The resulting F-score of 89.60% shows the increase obtained from using gold-standard GRs generated from CCGbank rather than the CCGbank dependencies themselves (for which the F-score was 85.20%).

Another difference between DepBank and CCGbank is that DepBank has been manually corrected, whereas CCGbank, including the test sections, has been produced semi-automatically from the Penn Treebank. There are some constructions in CCGbank—noun compounds being a prominent example—which are often incorrectly analyzed, simply because the required information is not in the Penn Treebank. Thus the evaluation on CCGbank overstates the accuracy of the parser, because it is tuned to produce

the output in CCGbank, including constructions where the analysis is incorrect. A similar comment would apply to other parsers evaluated on, and using grammars extracted from, the Penn Treebank.

A contribution of this section has been to highlight the difficulties associated with cross-formalism parser comparisons. Note that the difficulties are not unique to CCG, and many would apply to any cross-formalism comparison, especially with parsers using automatically extracted grammars. Parser evaluation has improved on the original PARSEVAL measures (Carroll, Briscoe, and Sanfilippo 1998), but the challenge still remains to develop a representation and evaluation suite which can be easily applied to a wide variety of parsers and formalisms.

## 12. Future Work

One of the key questions currently facing researchers in statistical parsing is how to adapt existing parsers to new domains. There is some experimental evidence showing that, perhaps not surprisingly, the performance of parsers trained on the WSJ Penn Treebank drops significantly when the parser is applied to domains outside of newspaper text (Gildea 2001; Lease and Charniak 2005). The difficulty is that developing new treebanks for each of these domains is infeasible. Developing the techniques to extract a CCG grammar from the Penn Treebank, together with the preprocessing of the Penn Treebank which was required, took a number of years; and developing the Penn Treebank itself also took a number of years.

Clark, Steedman, and Curran (2004) applied the parser described in this article to questions from the TREC Question Answering (QA) track. Because of the small number of questions in the Penn Treebank, the performance of the parser was extremely poor—well below that required for a working QA system. The novel idea in Clark, Steedman, and Curran was to create new training data from questions, but to annotate at the lexical category level only, rather than annotate with full derivations. The idea is that, because lexical categories contain so much syntactic information, adapting just the supertagger to the new domain, by training on the new question data, may be enough to obtain good parsing performance. This technique assumes that annotation at the lexical category level can be done relatively quickly, allowing rapid porting of the supertagger. We were able to annotate approximately 1,000 questions in around a week, which led to an accurate supertagger and, combined with the Penn Treebank parsing model, an accurate parser of questions.

There are ways in which this porting technique can be extended. For example, we have developed a method for training the dependency model which requires lexical category data only (Clark and Curran 2006). Partial dependency structures are extracted from the lexical category sequences, and the training algorithm for the dependency model is extended to deal with partial data. Remarkably, the accuracy of the dependency model trained on data derived from lexical category sequences alone is only 1.3% labeled F-score less than the full data model. This result demonstrates the significant amount of syntactic information encoded in the lexical categories. Future work will look at applying this method to biomedical text.

We have shown how using automatically assigned POS tags reduces the accuracy of the supertagger and parser. In Curran, Clark, and Vadas (2006) we investigate using the multi-tagging techniques developed for the supertagger at the POS tag level. The idea is to maintain some POS tag ambiguity for later parts of the parsing process, using the tag probabilities to decide which tags to maintain. We were able to reduce the drop

in supertagger accuracy by roughly one half. Future work will also look at maintaing the POS tag ambiguity through to the parsing stage.

Currently we do not use the probabilities assigned to the lexical categories by the supertagger as part of the parse selection process. These scores could be incorporated as real-valued features, or as auxiliary functions, as in Johnson and Riezler (2000). We would also like to investigate using the generative model of Hockenmaier and Steedman (2002b) in a similar way. Using a generative model's score as a feature in a discriminative framework has been beneficial for reranking approaches (Collins and Koo 2005). Because the generative model uses local features similar to those in our log-linear models, it could be incorporated into the estimation and decoding processes without the need for reranking.

One way of improving the accuracy of a supertagger is to use the parser to provide large amounts of additional training data, by taking the lexical categories chosen by the parser as gold-standard training data. If enough unlabeled data is parsed, then the large volume can overcome the noise in the data (Steedman et al. 2002; Prins and van Noord 2003). We plan to investigate this idea in the context of our own parsing system.

## 13. Conclusion

This article has shown how to estimate a log-linear parsing model for an automatically extracted CCG grammar, on a very large scale. The techniques that we have developed, including the use of a supertagger to limit the size of the charts and the use of parallel estimation, could be applied to log-linear parsing models using other grammar formalisms. Despite memory requirements of up to 25 GB we have shown how a parallelized version of the estimation process can limit the estimation time to under three hours, resulting in a practical framework for parser development. One of the problems with modeling approaches which require very long estimation times is that it is difficult to test different configurations of the system, for example different feature sets. It may also not be possible to train or run the system on anything other than short sentences (Taskar et al. 2004).

The supertagger is a key component in our parsing system. It reduces the size of the charts considerably compared with naive methods for assigning lexical categories, which is crucial for practical discriminative training. The tight integration of the supertagger and parser enables highly efficient as well as accurate parsing. The parser is significantly faster than comparable parsers in the NLP literature. The supertagger we have developed can be applied to other lexicalized grammar formalisms.

Another contribution of the article is the development of log-linear parsing models for CCG. In particular, we have shown how to define a CCG parsing model which exploits all derivations, including nonstandard derivations. These nonstandard derivations are an integral part of the formalism, and we have answered the question of whether efficent estimation and parsing algorithms can be defined for models which use these derivations. We have also defined a new parsing algorithm for CCG which maximizes expected recall of predicate–argument dependencies. This algorithm, when combined with normal-form constraints, gives the highest parsing accuracy to date on CCGbank. We have also given competitive results on DepBank, outperforming a non-CCG parser (RASP), despite the considerable difficulties involved in evaluating on a gold standard which uses a different annotation scheme to the parser.

There has perhaps been a perception in the NLP community that parsing with CCG is necessarily ineffficient because of CCG's "spurious" ambiguity. We have

demonstrated, using state-of-the-art statistical models, that both accurate and highly efficient parsing is practical with CCG. Linguistically motivated grammars can now be used for large-scale NLP applications.[12]

## Appendix A

The following rules were selected primarily on the basis of frequency of occurrence in Sections 02–21 of CCGbank.

### Type-Raising Categories for *NP*, *PP* and *S*[*adj*]*NP*

| |
|---|
| $S/(S\backslash NP)$ |
| $(S\backslash NP)\backslash((S\backslash NP)/NP)$ |
| $((S\backslash NP)/NP)\backslash(((S\backslash NP)/NP)/NP)$ |
| $((S\backslash NP)/(S[to]\backslash NP))\backslash(((S\backslash NP)/(S[to]\backslash NP))/NP)$ |
| $((S\backslash NP)/PP)\backslash(((S\backslash NP)/PP)/NP)$ |
| $((S\backslash NP)/(S[adj]\backslash NP))\backslash(((S\backslash NP)/(S[adj]\backslash NP))/NP)$ |
| $(S\backslash NP)\backslash((S\backslash NP)/PP)$ |
| $(S\backslash NP)\backslash((S\backslash NP)/(S[adj]\backslash NP))$ |

### Unary Type-Changing Rules

The category on the left of the rule is rewritten bottom-up as the category on the right.

$$
\begin{aligned}
N &\Rightarrow NP \\
NP &\Rightarrow S/(S/NP) \\
NP &\Rightarrow NP/(NP\backslash NP) \\
S[dcl]\backslash NP &\Rightarrow NP\backslash NP \\
S[pss]\backslash NP &\Rightarrow NP\backslash NP \\
S[ng]\backslash NP &\Rightarrow NP\backslash NP \\
S[adj]\backslash NP &\Rightarrow NP\backslash NP \\
S[to]\backslash NP &\Rightarrow NP\backslash NP \\
(S[to]\backslash NP)/NP &\Rightarrow NP\backslash NP \\
S[dcl]/NP &\Rightarrow NP\backslash NP \\
S[dcl] &\Rightarrow NP\backslash NP \\
S[pss]\backslash NP &\Rightarrow (S\backslash NP)\backslash(S\backslash NP)
\end{aligned}
$$

12 The parser and supertagger, including source code, are freely available via the authors' Web pages.

$$S[ng]\backslash NP \Rightarrow (S\backslash NP)\backslash(S\backslash NP)$$
$$S[adj]\backslash NP \Rightarrow (S\backslash NP)\backslash(S\backslash NP)$$
$$S[to]\backslash NP \Rightarrow (S\backslash NP)\backslash(S\backslash NP)$$
$$S[ng]\backslash NP \Rightarrow (S\backslash NP)/(S\backslash NP)$$
$$S[pss]\backslash NP \Rightarrow S/S$$
$$S[ng]\backslash NP \Rightarrow S/S$$
$$S[adj]\backslash NP \Rightarrow S/S$$
$$S[to]\backslash NP \Rightarrow S/S$$
$$S[ng]\backslash NP \Rightarrow S\backslash S$$
$$S[dcl] \Rightarrow S\backslash S$$
$$S[ng]\backslash NP \Rightarrow NP$$
$$S[to]\backslash NP \Rightarrow N\backslash N$$

**Punctuation Rules**

A number of categories absorb a comma to the left, implementing the following schema:

$$, \quad X \Rightarrow X$$

The categories are as follows, where $S[*]$ matches an $S$ category with any or no feature:

$N$, $NP$, $S[*]$, $N/N$, $NP\backslash NP$, $PP\backslash PP$, $S/S$, $S\backslash S$, $S[*]\backslash NP$, $(S\backslash NP)\backslash(S\backslash NP)$, $(S\backslash NP)/(S\backslash NP)$, $((S\backslash NP)\backslash(S\backslash NP))\backslash((S\backslash NP)\backslash(S\backslash NP))$

Similarly, a number of categories absorb a comma to the right, implementing the following schema:

$$X \quad , \quad \Rightarrow X$$

The categories are as follows:

$N$, $NP$, $PP$, $S[dcl]$, $N/N$, $NP\backslash NP$, $S/S$, $S\backslash S$, $S[*]\backslash NP$, $(S[dcl]\backslash NP)/S$, $(S[dcl]\backslash S[dcl])\backslash NP$, $(S[dcl]\backslash NP)/NP$, $(S[dcl]\backslash NP)/PP$, $(NP\backslash NP)/(S[dcl]\backslash NP)$, $(S\backslash NP)\backslash(S\backslash NP)$, $(S\backslash NP)/(S\backslash NP)$

These are the categories which absorb a colon or semicolon to the left, in the same way as for the comma:

$N$, $NP$, $S[dcl]$, $NP\backslash NP$, $S[*]\backslash NP$, $(S\backslash NP)\backslash(S\backslash NP)$

These are the categories which absorb a colon or semicolon to the right:

$N$, $NP$, $PP$, $S[dcl]$, $NP\backslash NP$, $S/S$, $S[*]\backslash NP$, $(S[dcl]\backslash NP)/S[dcl]$, $(S\backslash NP)\backslash(S\backslash NP)$, $(S\backslash NP)/(S\backslash NP)$

These are the categories which absorb a period to the right:

$N$, $NP$, $S[*]$, $PP$, $NP\backslash NP$, $S\backslash S$, $S[*]\backslash NP$, $S[*]\backslash PP$, $(S[dcl]\backslash S[*])\backslash NP$, $(S\backslash NP)\backslash(S\backslash NP)$

These are the categories which absorb a round bracket to the left:

$N$, $NP$, $S[dcl]$, $NP\backslash NP$, $(S\backslash NP)\backslash(S\backslash NP)$

These are the categories which absorb a round bracket to the right:

$N$, $NP$, $S[dcl]$, $N\backslash N$, $N/N$, $NP\backslash NP$, $S[dcl]\backslash NP$, $S/S$, $S\backslash S$, $(N/N)\backslash(N/N)$, $(S\backslash NP)\backslash(S\backslash NP)$, $(S\backslash NP)/(S\backslash NP)$

There are some binary type-changing rules involving commas, where the two categories on the left are rewritten bottom-up as the category on the right:

$$\begin{aligned}
, \ NP &\Rightarrow (S\backslash NP)\backslash(S\backslash NP)\\
NP \ , &\Rightarrow S/S\\
S[dcl]/S[dcl] \ , &\Rightarrow S/S\\
S[dcl]/S[dcl] \ , &\Rightarrow (S\backslash NP)\backslash(S\backslash NP)\\
S[dcl]/S[dcl] \ , &\Rightarrow (S\backslash NP)/(S\backslash NP)\\
S[dcl]/S[dcl] \ , &\Rightarrow S\backslash S\\
S[dcl]\backslash S[dcl] \ , &\Rightarrow S/S
\end{aligned}$$

Finally, there is a comma coordination rule, and a semicolon coordination rule, represented by the following two schema:

$$, \ X \Rightarrow X\backslash X$$

$$; \ X \Rightarrow X\backslash X$$

The categories which instantiate the comma schema are as follows:

$N$, $NP$, $S[*]$, $N/N$, $NP\backslash NP$, $S[*]\backslash NP$, $(S\backslash NP)\backslash(S\backslash NP)$

The categories which instantiate the semicolon schema are as follows:

$NP$, $S[*]$, $S[*]\backslash NP$

**Other Rules**

There are two rules for combining sequences of noun phrases and sequences of declarative sentences:

$$NP \;\; NP \Rightarrow NP$$

$$S[dcl] \;\; S[dcl] \Rightarrow S[dcl]$$

Finally, there are some coordination constructions in the original Penn Treebank which were difficult to convert into CCGbank analyses, for which the following rule is used:

$$\text{conj} \;\; N \;\; \Rightarrow N$$



**Appendix B**

The annotation in the *markedup* file for some of the most frequent categories in CCGbank is shown in Section 11.1. The annotation provides information about heads and dependencies, and also the mapping from CCG dependencies to the GRs in DepBank.

The first line after the unmarked lexical category gives the number of dependency relations plus the category annotated with head and dependency information. Variables in curly brackets indicate heads, with '_' used to denote the word associated with the lexical category. For example, if the word *buys* is assigned the transitive verb category $((S[dcl]\{\_\}\backslash NP\{Y\}\langle 1\rangle)\{\_\}/NP\{Z\}\langle 2\rangle)\{\_\}$, then the head on the resulting $S[dcl]$ is *buys*. Co-indexing of variables allows head passing; for example, in the relative pronoun category $((NP\{Y\}\backslash NP\{Y\}\langle 1\rangle)\{\_\}/(S[dcl]\{Z\}\langle 2\rangle\backslash NP\{Y^*\})\{Z\})\{\_\}$, the head of the resulting *NP* is taken from the *NP* which is modified to the left, and this head also becomes the subject of the verb phrase to the right. So in *the man who owns the company*, the subject of *owns* is *man*.

Numbers in angled brackets indicate dependency relations. For example, in the nominal modifier category $(N\{Y\}/N\{Y\}\langle 1\rangle)\{\_\}$, there is one dependency between the modifier and the modifiee. Long-range dependencies are indicated by marking head variables with $*$. The $*$ in the relative pronoun category indicates that when the $Y$ variable unifies with a lexical item, this creates a long-range subject dependency.

Some categories have a second head and dependency annotation, indicated with a *!*. This is used to produce the DepBank GRs. For example, the relative pronoun category has a second annotation which results in *who* being the subject of *owns* in *the man who owns the company*, rather than *man*, because this is consistent with DepBank. The first annotation is consistent with CCGbank.

The remaining lines in a category entry give the CCG dependencies to GRs mapping, described in Section 11.1.

```
N/N
  1 (N{Y}/N{Y}<1>){_}
  1 ncmod _ %f %l

NP[nb]/N
  1 (NP[nb]{Y}/N{Y}<1>){_}
  1 det %f %l
```

```
(NP\NP)/NP
  2 ((NP{Y}\NP{Y}<1>){_}/NP{Z}<2>){_}
  1 ncmod _ %f %l
  2 dobj %l %f

((S\NP)\(S\NP))/NP
  2 (((S[X]{Y}\NP{Z}){Y}\(S[X]{Y}<1>\NP{Z}){Y}){_}/NP{W}<2>){_}
  1 ncmod _ %f %l
  2 dobj %l %f

PP/NP
  1 (PP{_}/NP{Y}<1>){_}
  1 dobj %l %f

(S[dcl]\NP)/NP
  2 ((S[dcl]{_}\NP{Y}<1>){_}/NP{Z}<2>){_}
  1 ncsubj %l %f _
  2 xcomp _ %l %f =be
  2 dobj %l %f

(S\NP)\(S\NP)
  1 ((S[X]{Y}\NP{Z}){Y}\(S[X]{Y}<1>\NP{Z}){Y}){_}
  1 ncmod _ %f %l

(S[b]\NP)/NP
  2 ((S[b]{_}\NP{Y}<1>){_}/NP{Z}<2>){_}
  1 ncsubj %l %f _
  2 xcomp _ %l %f =be
  2 dobj %l %f

(S[to]\NP)/(S[b]\NP)
  2 ((S[to]{_}\NP{Z}<1>){_}/(S[b]{Y}<2>\NP{Z*}){Y}){_}
  1 ignore
  2 ignore

(S[dcl]\NP)/(S[b]\NP)
  2 ((S[dcl]{_}\NP{Y}<1>){_}/(S[b]{Z}<2>\NP{Y*}){Z}){_}
  ! ((S[dcl]{Z}\NP{Y}<1>){Z}/(S[b]{Z}<2>\NP{Y*}){Z}){_}
  1 ignore =aux
  1 ncsubj %l %f _
  2 aux %f %l =aux
  2 xcomp _ %l %f

(NP[nb]/N)\NP
  2 ((NP[nb]{Y}/N{Y}<1>){_}\NP{Z}<2>){_}
  1 ncmod poss %f %2
  2 ignore

S[adj]\NP
  1 (S[adj]{_}\NP{Y}<1>){_}
  1 ignore

S[pss]\NP
  1 (S[pss]{_}\NP{Y}<1>){_}
  1 ncsubj %l %f obj

(N/N)/(N/N)
  1 ((N{Y}/N{Y}){Z}/(N{Y}/N{Y}){Z}<1>){_}
```

```
  1 ncmod _ %f %l
(S[ng]\NP)/NP
  2 ((S[ng]{_}\NP{Y}<1>){_}/NP{Z}<2>){_}
  1 ncsubj %l %f _
  2 dobj %l %f

(S\NP)/(S\NP)
  1 ((S[X]{Y}\NP{Z}){Y}/(S[X]{Y}<1>\NP{Z}){Y}){_}
  1 ncmod _ %f %l

(S[dcl]\NP)/S[dcl]
  2 ((S[dcl]{_}\NP{Y}<1>){_}/S[dcl]{Z}<2>){_}
  1 ncsubj %l %f _
  2 ccomp _ %l %f

S[dcl]\NP
  1 (S[dcl]{_}\NP{Y}<1>){_}
  1 ncsubj %l %f _

(S[dcl]\NP)/(S[pt]\NP)
  2 ((S[dcl]{_}\NP{Y}<1>){_}/(S[pt]{Z}<2>\NP{Y*}){Z}){_}
  ! ((S[dcl]{Z}\NP{Y}<1>){Z}/(S[pt]{Z}<2>\NP{Y*}){Z}){_}
  1 ignore =aux
  1 ncsubj %l %f _
  2 aux %f %l =aux
  2 xcomp _ %l %f

S/S
  1 (S[X]{Y}/S[X]{Y}<1>){_}
  1 ncmod _ %f %l

(NP\NP)/(S[dcl]\NP)
  2 ((NP{Y}\NP{Y}<1>){_}/(S[dcl]{Z}<2>\NP{Y*}){Z}){_}
  ! ((NP{Y}\NP{Y}<1>){_}/(S[dcl]{Z}<2>\NP{_}){Z}){_}
  1 cmod %l %f %2
  2 ignore
```

## Acknowledgments

Much of this work was carried out at the University of Edinburgh's School of Informatics as part of Mark Steedman's EPSRC grant GR/M96889. We are immensely grateful to Mark for his guidance and advice during that time, and for allowing us the freedom to pursue the approach taken in this article. We are also grateful to Julia Hockenmaier for the use of CCGbank, the resource which made this work possible. Many of the ideas in this article emerged out of discussions with Mark and Julia. Jason Baldridge, Johan Bos, David Chiang, Claire Grover, Frank Keller, Yuval Krymolowski, Alex Lascarides, Mirella Lapata, Yusuke Miyao, Miles Osborne, Stefan Riezler, and Bonnie Webber have all given useful feedback. Thanks to Ted Briscoe, Rebecca Watson, and Stephen Pulman for help with the GRs, and thanks to Rebecca for carrying out the GR evaluation. We have also benefited from reviewers' comments on this article and various conference papers related to this work, and from the feedback of audiences at the universities of Cambridge, Edinburgh, Johns Hopkins, Oxford, Sheffield, and Sussex. While at Edinburgh, James Curran was funded by a Commonwealth scholarship and a University of Sydney traveling scholarship. He is currently supported by the Australian Research Council under Discovery Project DP0453131.

## References

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

Baldridge, Jason. 2002. *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. Ph.D. thesis, Edinburgh University, Edinburgh, UK.

Baldridge, Jason and Geert-Jan Kruijff. 2003. Multi-modal Combinatory Categorial Grammar. In *Proceedings of the 10th Meeting of the EACL*, pages 211–218, Budapest, Hungary.

Bangalore, Srinivas and Aravind Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265.

Bar-Hillel, Yehoshua. 1953. A quasi-arithmetical notation for syntactic description. *Language*, 29:47–58.

Borthwick, Andrew. 1999. *A Maximum Entropy Approach to Named Entity Recognition*. Ph.D. thesis, New York University, New York.

Bos, Johan, Stephen Clark, Mark Steedman, James R. Curran, and Julia Hockenmaier. 2004. Wide-coverage semantic representations from a CCG parser. In *Proceedings of COLING-04*, pages 1240–1246, Geneva, Switzerland.

Briscoe, Ted. 2006. An introduction to tag sequence grammars and the RASP system parser. Technical Report UCAM-CL-TR-662, University of Cambridge Computer Laboratory.

Briscoe, Ted and John Carroll. 2002. Robust accurate statistical annotation of general text. In *Proceedings of the 3rd LREC Conference*, pages 1499–1504, Las Palmas, Gran Canaria.

Briscoe, Ted and John Carroll. 2006. Evaluating the accuracy of an unlexicalized statistical parser on the PARC DepBank. In *Proceedings of the Poster Session of the Joint Conference of the International Committee on Computational Linguistics and the Association for Computational Linguistics (COLING/ACL-06)*, Sydney, Australia.

Briscoe, Ted, John Carroll, and Rebecca Watson. 2006. The second release of the RASP system. In *Proceedings of the Interactive Demo Session of the Joint Conference of the International Committee on Computational Linguistics and the Association for Computational Linguistics (COLING/ACL-06)*, pages 77–80, Sydney, Australia.

Buchholz, Sabine, Jorn Veenstra, and Walter Daelemans. 1999. Cascaded grammatical relation assignment. In *Proceedings of EMNLP/VLC-99*, pages 239–246, University of Maryland, College Park, MD.

Burke, Michael, Aoife Cahill, Ruth O'Donovan, Josef van Genabith, and Andy Way. 2004. Large-scale induction and evaluation of lexical resources from the Penn-II Treebank. In *Proceedings of the 42nd Meeting of the ACL*, pages 367–374, Barcelona, Spain.

Cahill, Aoife, Michael Burke, Ruth O'Donovan, Josef van Genabith, and Andy Way. 2004. Long-distance dependency resolution in automatically acquired wide-coverage PCFG-based LFG approximations. In *Proceedings of the 42nd Meeting of the ACL*, pages 320–327, Barcelona, Spain.

Carroll, John, Ted Briscoe, and Antonio Sanfilippo. 1998. Parser evaluation: A survey and a new proposal. In *Proceedings of the 1st LREC Conference*, pages 447–454, Granada, Spain.

Charniak, Eugene. 1997. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 598–603, Menlo Park, CA.

Charniak, Eugene. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st Meeting of the NAACL*, pages 132–139, Seattle, WA.

Chen, John, Srinivas Bangalore, Michael Collins, and Owen Rambow. 2002. Reranking an N-gram supertagger. In *Proceedings of the TAG+ Workshop*, pages 259–268, Venice, Italy.

Chen, John and K. Vijay-Shanker. 2000. Automated extraction of TAGS from the Penn Treebank. In *Proceedings of IWPT 2000*, Trento, Italy.

Chen, Stanley and Ronald Rosenfeld. 1999. A Gaussian prior for smoothing maximum entropy models. Technical Report CMU-CS-99-108, Carnegie Mellon University, Pittsburgh, PA.

Chiang, David. 2000. Statistical parsing with an automatically-extracted Tree Adjoining Grammar. In *Proceedings of the 38th Meeting of the ACL*, pages 456–463, Hong Kong.

Chiang, David. 2003. Mildly context sensitive grammars for estimating maximum entropy models. In *Proceedings of the 8th Conference on Formal Grammar*, Vienna, Austria.

Clark, Stephen. 2002. A supertagger for Combinatory Categorial Grammar. In *Proceedings of the TAG+ Workshop*, pages 19–24, Venice, Italy.

Clark, Stephen and James R. Curran. 2003. Log-linear models for wide-coverage CCG parsing. In *Proceedings of the EMNLP Conference*, pages 97–104, Sapporo, Japan.

Clark, Stephen and James R. Curran. 2004a. The importance of supertagging for wide-coverage CCG parsing. In

*Proceedings of COLING-04*, pages 282–288, Geneva, Switzerland.
Clark, Stephen and James R. Curran. 2004b. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Meeting of the ACL*, pages 104–111, Barcelona, Spain.
Clark, Stephen and James R. Curran. 2006. Partial training for a lexicalized-grammar parser. In *Proceedings of the Human Language Technology Conference and the Annual Meeting of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL'06)*, pages 144–151, New York.
Clark, Stephen and James R. Curran. 2007. Formalism-independent parser evaluation with CCG and DepBank. In *Proceedings of the 45th Meeting of the ACL*, Prague, Czech Republic.
Clark, Stephen, Julia Hockenmaier, and Mark Steedman. 2002. Building deep dependency structures with a wide-coverage CCG parser. In *Proceedings of the 40th Meeting of the ACL*, pages 327–334, Philadelphia, PA.
Clark, Stephen, Mark Steedman, and James R. Curran. 2004. Object-extraction and question-parsing using CCG. In *Proceedings of the EMNLP Conference*, pages 111–118, Barcelona, Spain.
Collins, Michael. 1996. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Meeting of the ACL*, pages 184–191, Santa Cruz, CA.
Collins, Michael. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4):589–637.
Collins, Michael and James Brooks. 1995. Prepositional phrase attachment through a backed-off model. In *Proceedings of the 3rd Workshop on Very Large Corpora*, pages 27–38, Cambridge, MA.
Collins, Michael and Terry Koo. 2005. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–69.
Collins, Michael and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Meeting of the ACL*, pages 111–118, Barcelona, Spain.
Crouch, Richard, Ronald M. Kaplan, Tracy H. King, and Stefan Riezler. 2002. A comparison of evaluation metrics for a broad-coverage stochastic parser. In *Proceedings of the LREC Beyond PARSEVAL workshop*, pages 67–74, Las Palmas, Spain.
Curran, James R. and Stephen Clark. 2003. Investigating GIS and smoothing for maximum entropy taggers. In *Proceedings of the 10th Meeting of the EACL*, pages 91–98, Budapest, Hungary.
Curran, James R., Stephen Clark, and David Vadas. 2006. Multi-tagging for lexicalized-grammar parsing. In *Proceedings of the Joint Conference of the International Committee on Computational Linguistics and the Association for Computational Linguistics (COLING/ACL-06)*, pages 697–704, Sydney, Australia.
Curry, Haskell B. and Robert Feys. 1958. *Combinatory Logic: Vol. I*. Amsterdam, The Netherlands.
Darroch, J. N. and D. Ratcliff. 1972. Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, 43(5):1470–1480.
Della Pietra, Stephen, Vincent Della Pietra, and John Lafferty. 1997. Inducing features of random fields. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 19(4):380–393.
Dienes, Peter and Amit Dubey. 2003. Deep syntactic processing by combining shallow methods. In *Proceedings of the 41st Meeting of the ACL*, pages 431–438, Sapporo, Japan.
Eisner, Jason. 1996. Efficient normal-form parsing for Combinatory Categorial Grammar. In *Proceedings of the 34th Meeting of the ACL*, pages 79–86, Santa Cruz, CA.
Geman, Stuart and Mark Johnson. 2002. Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *Proceedings of the 40th Meeting of the ACL*, pages 279–286, Philadelphia, PA.
Gildea, Daniel. 2001. Corpus variation and parser performance. In *2001 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 167–172, Pittsburgh, PA.
Goodman, Joshua. 1996. Parsing algorithms and metrics. In *Proceedings of the 34th Meeting of the ACL*, pages 177–183, Santa Cruz, CA.
Goodman, Joshua. 1997. Probabilistic feature grammars. In *Proceedings of the International Workshop on Parsing Technologies*, Cambridge, MA.
Gropp, W., E. Lusk, N. Doss, and A. Skjellum. 1996. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828.

Hockenmaier, Julia. 2003a. *Data and Models for Statistical Parsing with Combinatory Categorial Grammar*. Ph.D. thesis, University of Edinburgh, Edinburgh, UK.
Hockenmaier, Julia. 2003b. Parsing with generative models of predicate-argument structure. In *Proceedings of the 41st Meeting of the ACL*, pages 359–366, Sapporo, Japan.
Hockenmaier, Julia and Mark Steedman. 2002a. Acquiring compact lexicalized grammars from a cleaner treebank. In *Proceedings of the Third LREC Conference*, pages 1974–1981, Las Palmas, Spain.
Hockenmaier, Julia and Mark Steedman. 2002b. Generative models for statistical parsing with Combinatory Categorial Grammar. In *Proceedings of the 40th Meeting of the ACL*, pages 335–342, Philadelphia, PA.
Johnson, Mark. 1998. PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632.
Johnson, Mark. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of the 40th Meeting of the ACL*, pages 136–143, Philadelphia, PA.
Johnson, Mark, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic 'unification-based' grammars. In *Proceedings of the 37th Meeting of the ACL*, pages 535–541, University of Maryland, College Park, MD.
Johnson, Mark and Stefan Riezler. 2000. Exploiting auxiliary distributions in stochastic unification-based grammars. In *Proceedings of the 1st Meeting of the NAACL*, Seattle, WA.
Kaplan, Ron, Stefan Riezler, Tracy H. King, John T. Maxwell, III, Alexander Vasserman, and Richard Crouch. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proceedings of the Human Language Technology Conference and the 4th Meeting of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL'04)*, pages 97–104, Boston, MA.
Kasami, J. 1965. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA.
King, Tracy H., Richard Crouch, Stefan Riezler, Mary Dalrymple, and Ronald M. Kaplan. 2003. The PARC 700 Dependency Bank. In *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora*, Budapest, Hungary.
Koeling, Rob. 2000. Chunking with maximum entropy models. In *Proceedings of the CoNLL Workshop 2000*, pages 139–141, Lisbon, Portugal.
Lafferty, John, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, pages 282–289, Williams College, Williamstown, MA.
Lari, K. and S. J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4(1):35–56.
Lease, Matthew and Eugene Charniak. 2005. Parsing biomedical literature. In *Proceedings of the Second International Joint Conference on Natural Language Processing (IJCNLP-05)*, Jeju Island, Korea.
Levy, Roger and Christopher Manning. 2004. Deep dependencies from context-free statistical parsers: Correcting the surface dependency approximation. In *Proceedings of the 41st Meeting of the ACL*, pages 328–335, Barcelona, Spain.
Malouf, Robert. 2002. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the Sixth Workshop on Natural Language Learning*, pages 49–55, Taipei, Taiwan.
Malouf, Robert and Gertjan van Noord. 2004. Wide coverage parsing with stochastic attribute value grammars. In *Proceedings of the IJCNLP-04 Workshop: Beyond Shallow Analyses—Formalisms and Statistical Modeling for Deep Analyses*, Hainan Island, China.
Marcus, Mitchell, Beatrice Santorini, and Mary Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.
Miyao, Yusuke, Takashi Ninomiya, and Jun'ichi Tsujii. 2004. Corpus-oriented grammar development for acquiring a head-driven phrase structure grammar from the Penn Treebank. In *Proceedings of the First International Joint Conference on Natural Language Processing (IJCNLP-04)*, pages 684–693, Hainan Island, China.
Miyao, Yusuke and Jun'ichi Tsujii. 2002. Maximum entropy estimation for feature forests. In *Proceedings of the*

*Human Language Technology Conference*, San Diego, CA.

Miyao, Yusuke and Jun'ichi Tsujii. 2003a. A model of syntactic disambiguation based on lexicalized grammars. In *Proceedings of the Seventh Conference on Natural Language Learning (CoNLL)*, pages 1–8, Edmonton, Canada.

Miyao, Yusuke and Jun'ichi Tsujii. 2003b. Probabilistic modeling of argument structures including non-local dependencies. In *Proceedings of the Conference on Recent Advances in Natural Language Processing (RANLP)*, pages 285–291, Borovets, Bulgaria.

Miyao, Yusuke and Jun'ichi Tsujii. 2004. Deep linguistic analysis for the accurate identification of predicate-argument relations. In *Proceedings of COLING-2004*, pages 1392–1397, Geneva, Switzerland.

Miyao, Yusuke and Jun'ichi Tsujii. 2005. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proceedings of the 43rd meeting of the ACL*, pages 83–90, University of Michigan, Ann Arbor.

Moortgat, Michael. 1997. Categorial type logics. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*. Elsevier, Amsterdam, and The MIT Press, Cambridge, MA, pages 93–177.

Nasr, Alexis and Owen Rambow. 2004. Supertagging and full parsing. In *Proceedings of the TAG+7 Workshop*, Vancouver, Canada.

Nocedal, Jorge and Stephen J. Wright. 1999. *Numerical Optimization*. Springer, New York, NY.

Osborne, Miles. 2000. Estimation of stochastic attribute-value grammars using an informative sample. In *Proceedings of the 18th International Conference on Computational Linguistics*, pages 586–592, Saarbrücken, Germany.

Osborne, Miles and Ted Briscoe. 1997. Learning stochastic categorial grammars. In *Proceedings of the ACL CoNLL Workshop*, pages 80–87, Madrid, Spain.

Palmer, Martha, Dan Gildea, and Paul Kingsbury. 2005. The Proposition Bank: A corpus annotated with semantic roles. *Computational Linguistics*, 31(1):71–105.

Preiss, Judita. 2003. Using grammatical relations to compare parsers. In *Proceedings of the 10th Meeting of the EACL*, pages 291–298, Budapest, Hungary.

Prins, Robbert and Gertjan van Noord. 2003. Reinforcing parser preferences through tagging. *Traitement Automatique des Langues*, 44(3):121–139.

Ratnaparkhi, Adwait. 1996. A maximum entropy part-of-speech tagger. In *Proceedings of the EMNLP Conference*, pages 133–142, Philadelphia, PA.

Ratnaparkhi, Adwait. 1998. *Maximum Entropy Models for Natural Language Ambiguity Resolution*. Ph.D. thesis, University of Pennsylvania.

Ratnaparkhi, Adwait. 1999. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34(1–3):151–175.

Ratnaparkhi, Adwait, Salim Roukos, and Todd Ward. 1994. A maximum entropy model for parsing. In *Proceedings of the International Conference on Spoken Language Processing*, pages 803–806, Yokohama, Japan.

Riezler, Stefan, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell, III, and Mark Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the 40th Meeting of the ACL*, pages 271–278, Philadelphia, PA.

Sagae, Kenji and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the 9th International Workshop on Parsing Technologies*, pages 125–132, Vancouver, Canada.

Sarkar, Anoop and Aravind Joshi. 2003. Tree-adjoining grammars and its application to statistical parsing. In Rens Bod, Remko Scha, and Khalil Sima'an, editors, *Data-oriented parsing*. CSLI Publications, Stanford, CA.

Sha, Fei and Fernando Pereira. 2003. Shallow parsing with conditional random fields. In *Proceedings of the HLT/NAACL conference*, pages 213–220, Edmonton, Canada.

Steedman, Mark. 1996. *Surface Structure and Interpretation*. The MIT Press, Cambridge, MA.

Steedman, Mark. 2000. *The Syntactic Process*. The MIT Press, Cambridge, MA.

Steedman, Mark, Steven Baker, Stephen Clark, Jeremiah Crim, Julia Hockenmaier, Rebecca Hwa, Miles Osborne, Paul Ruhlen, and Anoop Sarkar. 2002. Semi-supervised training for statistical parsing: Final report. Technical Report CLSP WS-02, Center for Language and Speech Processing, Johns Hopkins University, Baltimore, MD.

Taskar, B., D. Klein, M. Collins, D. Koller, and C. Manning. 2004. Max-margin parsing. In *Proceedings of the EMNLP Conference*, pages 1–8, Barcelona, Spain.

Toutanova, Kristina, Christopher Manning, Stuart Shieber, Dan Flickinger, and Stephan Oepen. 2002. Parse disambiguation for a rich HPSG grammar. In *Proceedings of the First Workshop on Treebanks and Linguistic Theories*, pages 253–263, Sozopol, Bulgaria.

Toutanova, Kristina, Penka Markova, and Christopher Manning. 2004. The leaf projection path view of parse trees: Exploring string kernels for HPSG parse selection. In *Proceedings of the EMNLP conference*, pages 166–173, Barcelona, Spain.

Watkinson, Stephen and Suresh Manandhar. 2001. Acquisition of large categorial grammar lexicons. In *Proceedings of the Conference of the Pacific Association for Computational Linguistics (PACLING-01)*, Kitakyushu, Japan.

Watson, Rebecca, John Carroll, and E. J. Briscoe. 2005. Efficient extraction of grammatical relations. In *Proceedings of the 9th International Workshop on Parsing Technologies*, pages 160–170, Vancouver, Canada.

Wood, Mary McGee. 1993. *Categorial Grammars*. Routledge, London.

Xia, Fei, Martha Palmer, and Aravind Joshi. 2000. A uniform method of grammar extraction and its applications. In *Proceedings of the EMNLP Conference*, pages 53–62, Hong Kong.

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

Zettlemoyer, Luke S. and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*, pages 658–666, Edinburgh, UK.