

# Improving the complement/adjunct distinction in CCGbank

Matthew Honnibal and James R. Curran

School of Information Technologies

University of Sydney

NSW 2006, Australia

{mhonn, james}@it.usyd.edu.au

## Abstract

One of the challenges of adapting the Penn Treebank for a specific formalism is that the target annotation often requires information represented imperfectly or not at all in the original corpus. When this occurs, the information must either be guessed with heuristics, or annotated manually. Recently, a third option has become available, due to the release of resources that supplement the Penn Treebank, such as the predicate-argument annotation provided by PropBank.

In this paper, we discuss how such supplementary resources can be used to correct resources acquired from the Penn Treebank before they were available. We describe how we use the PropBank annotation to correct the complement and adjunct labels in CCGbank, producing cleaner, more consistent annotation. We also demonstrate that the new corpus can still be accurately reproduced by a statistical parser.

## 1 Introduction

The Penn Treebank's (PTB) annotation scheme was designed to make the corpus as widely useful as possible, but also to avoid drawing distinctions which the annotators found difficult to apply (Marcus et al., 1994). The corpus can be adapted to suit other formalisms, but problems will be encountered if the new annotation scheme requires information not represented in the original trees.

For instance, combinatory categorial grammar (CCG) requires binary branching trees, and distinguishes complements from adjuncts. The Penn Treebank does not describe the internal structure of noun phrases, and does not consistently apply complement and adjunct labels. Hockenmaier

and Steedman (2005) created a CCG version of the PTB, but had to rely on heuristics in cases such as these, introducing noise into the resulting corpus.

Since the publication of the Penn Treebank, several projects have supplemented it by adding some of this missing information. Vadas and Curran (2007), for instance, have manually bracketed the noun phrases in the corpus whose structure was ambiguous. Another example is PropBank (Palmer et al., 2005), which adds predicate-argument annotation to the Penn Treebank. Recently, Shen et al. (2007) used PropBank to create a novel LTAG corpus from the Penn Treebank, LTAG-spinal.

In this paper, we show that such supplementary resources can be used to reform a corpus acquired from the Penn Treebank, correcting noise introduced by conversion heuristics. We use the semantic role labels in PropBank as the basis for the syntactic distinction between complements and adjuncts CCG requires. In order to do this, we first create a version of PropBank's stand-off annotation that refers to nodes in CCGbank, rather than nodes in the Penn Treebank. We then find arguments whose CCGbank label is inconsistent with their PropBank label, and apply changes to the CCG derivation.

When making these changes, we must avoid producing faulty derivations. Most CCG productions are licensed by a small set of general combinatory rules. To avoid creating productions unlicensed by these rules, we must propagate changes down the derivation tree. The rules for doing this, and for selecting the most appropriate new node label, must be as robust to noise in PropBank and CCGbank as possible.

To evaluate the resulting corpus, we check how often the changes have produced invalid derivations, and how consistently the reformed distinction between complements and adjuncts can be drawn by a parser. Validation is performed by

checking whether each production in the sentence occurred in the original corpus, or is licensed by the general combinatory rules. We find that only 22 sentences fail this validation process – a tiny proportion of the 10,254 sentences we have changed. We also find that the new complement and adjunct distinction does not dramatically impact the accuracy of a C&C parser (Clark and Curran, 2007) trained and evaluated on the modified corpus.

## 2 The Importance of Accurate Complement/Adjunct Labels

‘Deep’ grammars, such as LFG, HPSG, CCG and LTAG, attempt to generate semantic analyses, instead of skeletal parses. This typically involves producing a predicate-argument structure or logical form of the sentence. By contrast, ‘shallow’ parsers pay little attention to the interface between grammar and semantics, leaving the semantic extraction as a task for some downstream process.

The distinction between a complement — a generally obligatory, core argument of the predicate — and an adjunct — an optional argument that provides some auxiliary information from a restricted set of categories — is one of the tasks of this semantic analysis. Since a statistical parser learns its distinctions from the examples it is given, the accuracy of a ‘deep’ parser’s analysis therefore depends to some extent on the accuracy of the complement and adjunct labels in its training corpus.

It is particularly useful for the corpus’s complement and adjunct labels to be drawn from a semantic analysis that includes more information than the parser provides. PropBank (Palmer et al., 2005) and FrameNet (Baker et al., 1998) are two examples of such analyses. If the parser is designed to replicate part of the annotation of one of these resources, it will then be easier to recover the rest of the information. For instance, if the argument structure of each verb in CCGbank is made to replicate the argument structure of each predicate in PropBank, the only missing information are the finer grained semantic role labels.

## 3 Combinatory Categorial Grammar

CCG (Steedman, 2000) is a linguistically motivated formalism that extends Categorial Grammar (Bar-Hillel, 1953) with a small set of operations, most of which were inspired by combinatory logic

(Curry and Feys, 1958). Each word is assigned a category, and the categories are then combined according to a small set of rules to form the derivation. In the original Categorial Grammar, there are only two combinatory rules, forward application and backward application:

$$X/Y \ Y \Rightarrow X \ (>) \quad (1)$$

$$Y \ X \backslash Y \Rightarrow X \ (<) \quad (2)$$

where  $X$  and  $Y$  denote categories (either basic or complex). Figure 1 shows a derivation using these two rules. The category of the main verb, *join*, can be regarded as a function from two *NP* arguments to a sentence. Categorial Grammar is context-free, and so does not adequately describe long-range dependencies. The extra combinators in CCG increase the generative power of the grammar to mildly context sensitive (Steedman, 2000).

Because every combinatory rule operates on exactly two categories, derivations are generally binary trees; although there are a few unary productions too. Unlike a phrase-structure grammar, production rules are not arbitrary pairings of parent and child labels: two categories can usually only be combined to form one parent, so if only one node label is changed the derivation will almost always be rendered invalid.

## 4 CCGbank

Hockenmaier and Steedman (2005) adapted the Penn Treebank into a CCG corpus, CCGbank. The project was similar to other theory-specific treebanks acquired from more general resources, such as Chen and Vijay-Shanker (2000)’s TAG corpus. The CCGbank annotation consists of a CCG derivations and corresponding dependencies for 96% of the sentences in the Penn Treebank. The derivations are binary trees that record the categories, but not the combinatory rules invoked.

One of the main difficulties in the adaptation this was that a CCG analysis often encodes more linguistic information than is present in the original tree, as CCG relies on distinctions which the Penn Treebank annotators found too costly to reliably annotate. For instance, Hockenmaier employs head-finding heuristics much like those described by Magerman (1995) to binarise noun phrases, since the internal structure of noun-phrases is often ambiguous in the Penn Treebank description. Another source of difficulty were

structures where the original analysis differed substantially from the desired CCG analysis. There are many constructions which receive non-traditional analyses in CCG, as the properties of the formalism present different possibilities. Extraction and co-ordination constructions are prominent examples of this.

Constituent function labels are another case where the CCG formalism offers a different analysis from a phrase-structure grammar. In the Penn Treebank, function labels are encoded as extra information on certain nodes. This information is difficult to encode in a phrase-structure grammar, so it is generally not used by parsers trained on the original treebank, and the function labels were not rigorously checked and corrected when the corpus was created (Bies, 1995).

In CCG, however, the valency of a verb is represented in its category, and so the distinction between complements and adjuncts is represented in the derivation. Adjuncts are categories of the form  $X \setminus X$  or  $X / X$ . That is, they are functions that take a category and return it unchanged, allowing unlimited other adjuncts to attach. Complements are usually atomic categories that are arguments of the verb. When CCGbank was created, complements and adjuncts were distinguished using the inconsistent function labels in the Penn Treebank, introducing noise into the corpus.

Several statistical parsers have been trained and evaluated on CCGbank. We use the C&C parser described by Clark and Curran (2007) to examine how the changes we have made have affected this use of the corpus.

## 5 PropBank

PropBank (Palmer et al., 2005) provides predicate-argument structure annotation for each sentence in the Penn Treebank. This annotation involves identifying each predicate in the sentence, and which constituents realise its arguments. The arguments then receive detailed labels that specify their semantic role, capturing generalisations that are complicated by surface syntactic constructions such as passivisation and ergativity.

The label scheme begins by describing each argument as either core or peripheral. Core arguments receive a numeric semantic role label, the interpretation of which is specific to each predicate. Peripheral arguments receive a label from a generic set. Peripheral arguments are adjuncts,

and are defined by the fact that they are optional, cannot become the subject of the predicate, and can be assigned a label from a small set generic to all predicates, such as direction, purpose and time.

PropBank is a semantic annotation layer, so the target of the argument labels may not be a single constituent. Annotation is provided in a stand-off format, and labels may refer to multiple nodes, and often include references to traced constituents. However, syntactic distinctions are given priority over semantic differences when determining the frame set of a particular predicate, in order to avoid making distinctions that are too fine grained.

## 6 Aligning CCGbank and PropBank

In order to use the PropBank annotation for CCGbank, we first align the two resources. First, we align the sentences and tokens in CCGbank with the original PTB text, so that the few sentences missing from CCGbank do not cause the PropBank indices to refer to the wrong sentences. Token alignment is necessary too, because CCGbank sentences do not include the PTB null elements. CCGbank also omits quotation marks.

Once we have the correct sentences and token indices, we must select a set of CCG nodes that correspond to the PTB nodes that realise an argument. Since we cannot expect the CCG derivation to match the structure of the original parse, we search for the smallest set of CCG nodes that cover all and only the tokens covered by the PTB nodes PropBank refers to.

To do this, we first collect the tokens covered by the PTB nodes that realise the PropBank argument. For each word, we move up the derivation until we find a node whose parent includes at least one token not in the set we are looking for. We then add this node to the set of CCG nodes that will realise the argument. If two or more tokens in the set have a common ancestor that does not span any tokens not in the set, that ancestor will be referred to instead of the tokens it yields.

## 7 Changing CCGbank

This section describes the process we used to find inconsistencies between PropBank and CCGbank, and how changes were applied.

### 7.1 Changing node labels

In a phrase-structure grammar, production rules are arbitrary and unbounded: there is no underlying

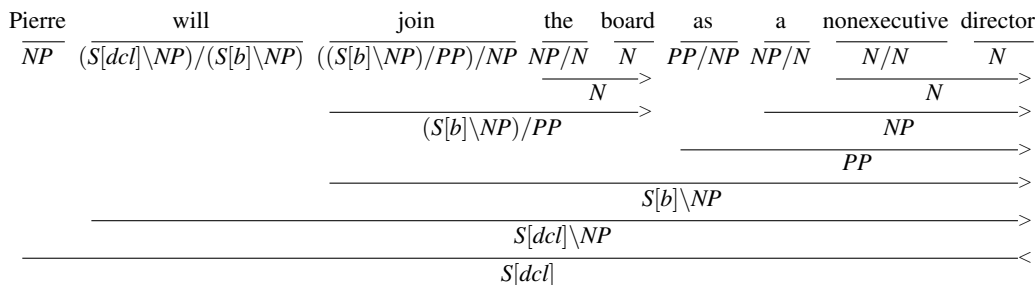


Figure 1: The original CCGbank derivation of the first sentence of the Penn Treebank.

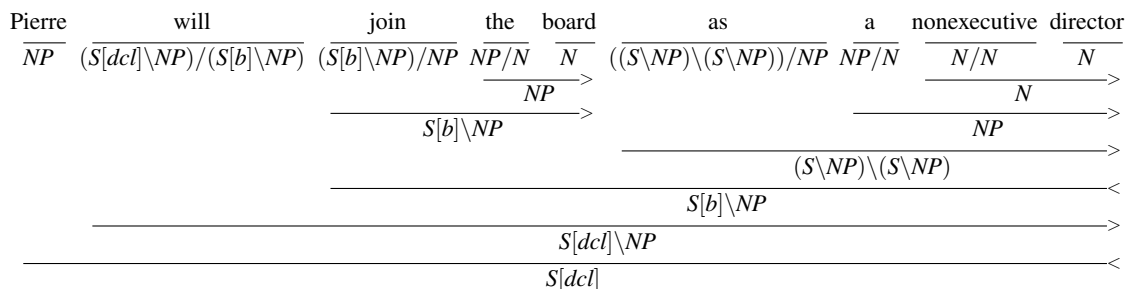


Figure 2: The altered derivation, with the *PP* argument changed to an adjunct.

ing logic behind why one symbol should be rewritten as the set of symbols that make up its children. As discussed in section 3, this is not true in CCG. We cannot simply replace one node label with another, or we will introduce an invalid production rule. When a node label is changed, the changes must be reflected in the surrounding tree.

The sibling of the node we are changing will always be the verb, as discussed in section 7.2. The new verb category will be produced by adding or deleting an argument, depending on whether we are changing an adjunct to a complement or vice versa. The rules for deciding the new sibling category are therefore given in sections 7.3 and 7.4.

We do this by finding instances of the old label in the subtree, and replacing them with the new label. We must replace the old category if it occurs as a whole label, a result, or an argument. Figure 3 shows a tree where the parent category is produced by using the forward application rule on the two children. This means that the parent category will be the result category of the left child, and the left child’s argument category will be the right child’s whole label.

Unary productions present a special case not covered by the rules discussed above. Unary productions are rules where the parent has only one child.

CCG does license unary type-raising rules, and Hockenmaier and Steedman (2005) discusses the

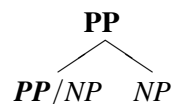


Figure 3: The parent category is the result category of the left child.

addition of several extra unary rules to simplify the category set. Other unary productions seem to be the result of noise. In the latter two cases, there is no obvious relationship between the parent category and the child category, as there would be with a binary production licensed by the grammar. This prevents us from using the rules discussed above to propagate the changes down the tree.

Figure 4 shows a tree which we would like to change to an argument, without introducing a new production rule  $NP \rightarrow N$ . If we simply change the parent category and not the child, we will probably be introducing a new rule into the grammar. Since we wish to avoid this, we instead collapse the unary production, moving the grand-children up to the node whose label we have just changed, before propagating the changes to them. Figure 5 shows the tree after we have applied this process.

## 7.2 Finding arguments

This section discusses how we identify inconsistencies between the PropBank and CCG argument structure annotation. We iterate through

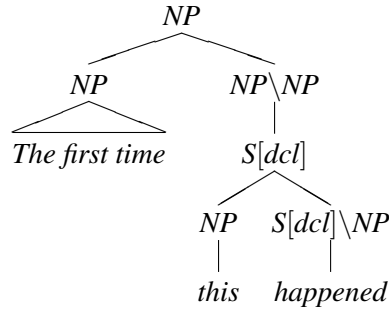


Figure 6: PropBank annotates ‘The first time’ as a peripheral argument of ‘happened’. We do not change cases like this.

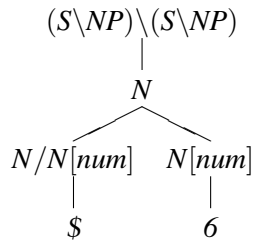


Figure 4: An adjunct with a unary production.

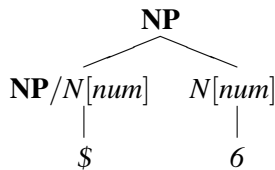


Figure 5: The tree relabelled as a complement, with the unary production collapsed and changes propagated through the subtree.

each PropBank argument, using several criteria to decide whether its CCG category ought to be changed.

The first criterion is that the PropBank argument should be realised by a single CCG constituent. This excludes 1.6% (6,271) of arguments from consideration. We are not interested in these arguments, because we are looking for cases where we can identify a simple inconsistency between the PropBank label and the CCG category. If there is not a single constituent, the two annotations disagree more fundamentally than our rules are designed to correct. Often, it would not even make sense to consider the PropBank argument as a syntactic constituent, such as the many cases where there is intervening text between the constituents that realise it.

Next, we check that the CCG category is consistent with the PropBank label. If the PropBank label is ARG<sub>M</sub>, the constituent should have a sen-

tential adjunct category like  $(S\backslash NP)\backslash(S\backslash NP)$ ,  $S/S$ , etc. Sentential adjunct categories are distinguished by having identical results and arguments, and having an innermost result category of  $S$ . If the argument has a numeric PropBank label, indicating that it is a subject or complement, its constituent should not have a CCG adjunct category unless it is the agent of a passive verb. 3.9% (15,686) of PropBank arguments have labels inconsistent with their CCG categories in this way.

The other criteria we consider are intended to guarantee that in the CCG derivation the argument attaches directly to the predicate, as either a CCG adjunct or a CCG argument. First, we check that the sibling of the constituent is a verb. The first criterion we use to do this is to check that its category’s innermost result is  $S$ . Next, we check that the category is not an adjunct, by checking whether its argument is identical to its result. We also do not wish to change or introduce leftward arguments, so we add an extra constraint rejecting arguments which occur before the verb they attach to. This constraint is very rarely required, rejecting a further 10 arguments only.

6% (946) of the 15,686 arguments with inconsistent PropBank and CCGbank annotations fail these criteria. The most common case where this criterion is not met is when a relative clause is attached to a noun that PropBank considers a peripheral argument. Figure 6 shows a tree fragment that exemplifies this. 62.2% (583) of the 936 cases described above are like this example.

Once we have established that the constituent is attached to a verb, we check that the head of that verb node is the predicate the PropBank entry refers to. The case this corrects is rare: only 0.7% (98) of the 14,750 remaining inconsistencies exhibit this problem. Nevertheless, catching this case greatly reduces the number of invalid deriva-

tions our changes produce.

### 7.3 Changing complements to adjuncts

Changing a complement to an adjunct involves changing the constituent’s label, the verb label, and updating the dependencies. The new constituent label is always  $(S \setminus NP) \setminus (S \setminus NP)$ , the standard non-fronted adjunct category. Our constraints ensure that the constituent occurs after the verb and that the verb category is not atomic.

As discussed above, the verb node is the sibling of the constituent. The new category of this constituent will be its current result, as we are simply changing the argument to an adjunct. This also ensures that the parent category will remain unchanged.

These changes alter which CCG rules are invoked in the derivation. In the simplest case, the verb and the constituent no longer combine by forward application of the verb category, but by backward application of the new adjunct. Figure 2 shows an example of this.

Occasionally, however, the new derivation may require backward composition, as the verb could still looking for rightward arguments. Table 1 lists the instances of the various combinatory rules in CCGbank derivations before and after our changes, showing that despite our initial concern, the new corpus actually contains slightly fewer productions validated by composition rules.

Having changed the derivation, we must update the dependencies it produces. We must delete one local dependency, and introduce one local dependency, as instead of the verb governing the argument, the argument is now an adjunct – a functor category that governs the verb.

We change 1,543 complements to adjuncts.

### 7.4 Changing adjuncts to complements

Changing an adjunct to a complement involves selecting the new label of the complement, adding it as an argument to the verb category, and updating the dependencies. Selecting the appropriate label for the complement can be tricky, as CCG categories reflect the constituent’s function, not its form. An *NP* functioning as an adjunct and a *PP* functioning as an adjunct will therefore both have the same category. The first heuristic we use deals with adjuncts formed by unary productions. In these cases, the adjunct node is essentially replaced by its child, as we simply need to undo the unary rule. Otherwise, we try to decide whether

| Rule             | Before    | After     |
|------------------|-----------|-----------|
| F. App.          | 674,487   | 686,209   |
| B. App.          | 227,552   | 217,621   |
| F. Comp.         | 4,834     | 4,835     |
| Gen. F. Comp     | 21        | 21        |
| F. X. Comp       | 51        | 51        |
| Gen. F. X. Comp  | 1,314     | 1,226     |
| B. Comp.         | 1,601     | 1,520     |
| Gen. B. Comp     | 237       | 238       |
| B. X. Comp.      | 13,582    | 11,841    |
| Gen. B. X. Comp. | 1,560     | 1,529     |
| Total valid      | 1,094,249 | 1,094,303 |
| Total invalid    | 5,243     | 5,189     |

Table 1: The frequency with which CCG rules are invoked to validate production rules before and after our changes. F.=Forwards, B.=Backwards, App.=Application, Comp.=Composition, X.=Crossing, Gen.=Generalised.

the complement is to be labelled *NP* or *PP* by checking whether its head is a noun, based on its part of speech tag.

Once we have the category of the complement, we can add it as an argument to the verb. The new verb category will be of the form  $X/Y$ , where  $X$  is the old verb category, and  $Y$  is the complement category.

Having changed the derivation, we must update the dependencies it produces. We must delete one local dependency, and introduce one local dependency, as instead of the complement governing the verb as an adjunct, the verb now governs the complement as an argument.

We change 13,256 adjuncts to complements. Many of these are arguments of verbs of price motion.

## 8 Validating the changes

Once we have changed the corpus, we wish to know whether our alterations have had an adverse effect. We approach this issue by asking how often our changes have produced derivations not licensed by the original grammar. Validation proceeds in two steps. First, we check whether each production in the new corpus occurred in the original. If a production did not occur in the original corpus, we check whether it is licensed by the general combinatory rules.

Novel productions are sometimes introduced when changing adjuncts to complements intro-

duces verb categories with novel argument structures. Other novel productions seem to be the result of the unpredictable interaction of our algorithm with noise in CCGbank.

In order to further examine these novel production rules, we check whether they are licensed by the general combinatory rules of CCG. If the derivation passes this test, the changes are preserved; if not the changes are reverted. Very few sentences fail this validation process: only 22 out of 10,254 changed.

It is also interesting to apply the two validation processes in the opposite order. This also examines how many derivations in the original corpus are licensed by the CCG combinatory rules. We considered all unary productions valid for the purpose of this evaluation. We also added a few extra rules that deal with punctuation. These rules effectively treat punctuation marks as having schematic categories like  $X/X$  and  $X\backslash X$ , allowing them to be ‘absorbed’ into any category to their left or right.

Table 1 describes how often the various rules were invoked to license productions in the corpus before and after changes. It also shows how many productions could not be validated by the rules alone. The table shows that despite our concerns, our changes have not actually introduced more composition productions into the corpus.

## 9 Parsing results

Table 2 presents the results from evaluating the C&C parser on section 23 of CCGbank, before and after changes. The results show that the parser performs slightly worse on the new corpus. There are several possible reasons for this.

One possibility is that the increase in complements in the new corpus simply makes the task harder. Because verb categories reflect the predicate’s argument structure, increasing the number of non-NP complements may make the supertagger’s task of selecting the correct verb category slightly more difficult. On the other hand, the small difference in performance may simply be due to a more even balance between complements and adjuncts in the new corpus, since prepositional phrases are usually adjuncts.

Another possible explanation for reduced performance is that the parameters used in the supertagger and parser, e.g. levels of ambiguity used in supertagging, have been optimised for the original CCGbank, and we are yet to repeat this process

for the new corpus.

## 10 Conclusion

Treebanks are expensive projects, and limited resources are available for their construction. This typically forces compromises to be made in the construction of a treebank, both in the information its annotation scheme is designed to represent, and how rigorously that annotation scheme is enforced.

One of the most common compromises made is to acquire a theory specific corpus from a more general one. This is a very effective strategy, but one that generally introduces some level of noise into the resulting corpus. One of the principle sources of such noise is distinctions made in the new annotation scheme that are only partially supported by the original corpus.

We have shown how new information can be incorporated into a corpus automatically acquired from the Penn Treebank, in order to mitigate this problem. As supplementary resources are released for the Penn Treebank, corpora calculated from it can be updated, thereby benefiting from these steady improvements.

Improving the training data with respect to this distinction may also improve parsing results. Clark and Curran (2007) report that the distinction between complements, adjuncts and NP qualifiers is one of the main sources of error. To some extent, this is unsurprising, as PP attachment is one of the most difficult distinctions to draw. However, the inconsistent complement and adjunct labels that CCGbank inherits from the Penn Treebank may well be a problem. Although our current results show little change in parsing performance, we believe future experimentation will see the benefit of making CCGbank more consistent.

We have demonstrated how the practical issues involved in reforming a problematic linguistic distinction can be overcome. We have described how the three resources required — CCGbank, the Penn Treebank, and PropBank — can be aligned; and how the PropBank labels can be used to correct the CCGbank derivations. The reformed complement and adjunct labels, which sometimes involve more subtle semantic distinctions drawn by the PropBank annotators, can be replicated by a parser almost as accurately as the original annotation, without tuning the parser to better handle the changes in the corpus.

| Model    | LP    | LR    | LF    | SENT  | UP    | UR    | UF    | CAT   | cov   |
|----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| Original | 85.75 | 84.88 | 85.31 | 32.28 | 92.53 | 91.59 | 92.05 | 93.19 | 99.27 |
| Modified | 84.31 | 82.91 | 83.61 | 29.24 | 92.53 | 90.99 | 91.75 | 93.00 | 99.22 |

Table 2: Results on Section 00 with both models using normal-form constraints

## 11 Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments. James Curran was funded under ARC Discovery grants DP0453131 and DP0665973.

## References

- Colin F. Baker, Charles J. Fillmore, and John B. Lowe. 1998. The Berkeley FrameNet project. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics*, pages 86–90. Montreal, Canada.
- Yehoshua Bar-Hillel. 1953. A quasi-arithmetical notation for syntactic description. *Language*, 29:47–58.
- A. Bies. 1995. Bracketing guidelines for Treebank II style. Penn Treebank Project.
- J. Chen and K. Vijay-Shanker. 2000. Automated extraction of tags from the penn treebank. In *Proceedings of the 6th International Workshop on Parsing Technologies*. Trento, Italy.
- Stephen Clark and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*. (to appear).
- Haskell B. Curry and Robert Feys. 1958. *Combinatory Logic: Vol. I*. Amsterdam, North Holland.
- Julia Hockenmaier and Mark Steedman. 2005. Ccgbank manual. Technical Report MS-CIS-05-09, University of Pennsylvania.
- David Magerman. 1995. Statistical decision-tree models for parsing. In *Proceedings of the 33rd Meeting of the ACL*, pages 276–283. Cambridge, MA.
- Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1994. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.
- Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106.
- Libin Shen, Lucas Champollion, and Aravind K. Joshi. 2007. LTAG-spinal and the treebank. *Language Resources and Evaluation*. to appear.
- Mark Steedman. 2000. *The Syntactic Process*. The MIT Press, Cambridge, MA.
- David Vadas and James R. Curran. 2007. Adding noun phrase structure to the penn treebank. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, pages 240–247. Prague, Czech Republic.