Using linguistic principles to recover empty categories

Richard CAMPBELL
Microsoft Research
One Microsoft Way
Redmond, WA 98052
USA
richcamp@microsoft.com

Abstract

This paper describes an algorithm for detecting empty nodes in the Penn Treebank (Marcus et al., 1993), finding their antecedents, and assigning them function tags, without access to lexical information such as valency. Unlike previous approaches to this task, the current method is not corpus-based, but rather makes use of the principles of early Government-Binding theory (Chomsky, 1981), the syntactic theory that underlies the annotation. Using the evaluation metric proposed by Johnson (2002), this approach outperforms previously published approaches on both detection of empty categories and antecedent identification, given either annotated input stripped of empty categories or the output of a parser. Some problems with this evaluation metric are noted and an alternative is proposed along with the results. The paper considers the reasons a principle-based approach to this problem should outperform corpus-based approaches, and speculates on the possibility of a hybrid approach.

1 Introduction

Many recent approaches to parsing (e.g. Charniak, 2000) have focused on labeled bracketing of the input string, ignoring aspects of structure that are not reflected in the string, such as phonetically null elements and long-distance dependencies, many of which provide important semantic information such as predicate-argument structure. In the Penn Treebank (Marcus et al., 1993), null elements, or empty categories, are used to indicate non-local dependencies, discontinuous constituents, and certain missing elements. Empty categories are coindexed with their antecedents in the same sentence. In addition, if a node has a particular grammatical function (such as subject) or semantic role (such as location), it has a function tag indicating that role; empty categories may also have function tags. Thus in the sentence below, *who* is coindexed with the empty category **T** in the embedded S; the function tag *SBJ* indicates that this empty category is the subject of that S:

$[_{\text{WHNP-1}}$ who] NP want $[_{\text{S}}$ $[_{\text{NP-SBJ-1}}$*T*] to VP]

Empty categories, with coindexation and function tags, allow a transparent reconstruction of predicate-argument structure not available from a simple bracketed string.

In addition to bracketing the input string, a fully adequate syntactic analyzer should also locate empty categories in the parse tree, identify their antecedents, if any, and assign them appropriate function tags. State-of-the-art statistical parsers (e.g. Charniak, 2000) typically provide a labeled bracketing only; i.e., a parse tree without empty categories. This paper describes an algorithm for inserting empty categories in such impoverished trees, coindexing them with their antecedents, and assigning them function tags. This is the first approach to include function tag assignment as part of the more general task of empty category recovery.

Previous approaches to the problem (Collins, 1997; Johnson, 2002; Dienes and Dubey, 2003a,b; Higgins, 2003) have all been learning-based; the primary difference between the present algorithm and earlier ones is that it is not learned, but explicitly incorporates principles of Government-Binding theory (Chomsky, 1981), since that theory underlies the annotation. The absence of rule-based approaches up until now is not motivated by the failure of such approaches in this domain; on the contrary, no one seems to have tried a rule-based approach to this problem. Instead it appears that there is an understandable predisposition against rule-based approaches, given the fact that data-driven, especially machine-learning, approaches have worked so much better in many other domains.[1]

Empty categories however seem different, in that, for the most part, their location and existence is determined, not by observable data, but by explicitly constructed linguistic principles, which

[1]Both Collins (1997: 19) and Higgins (2003: 100) are explicit about this predisposition.

were consciously used in the annotation; i.e., unlike overt words and phrases, which correspond to actual strings in the data, empty categories are in the data only because linguists doing the annotation put them there. This paper therefore explores a rule-based approach to empty category recovery, with two purposes in mind: first, to explore the limits of such an approach; and second, to establish a more realistic baseline for future (possibly data-driven or hybrid) approaches.

Although it does not seem likely that any application trying to glean semantic information from a parse tree would care about the exact string position of an empty category, the algorithm described here does try to insert empty categories in the correct position in the string. The main reason for this is to facilitate comparison with previous approaches to the problem, which evaluate accuracy by including such information. In Section 5, however, a revised evaluation metric is proposed that does not depend on string position per se.

Before proceeding, a note on terminology is in order. I use the term *detection* (of empty categories) for the insertion of a labeled empty category into the tree (and/or string), and the term *resolution* for the coindexation of the empty category with its antecedent(s), if any. The term *recovery* refers to the complete package: detection, resolution, and assignment of function tags to empty categories.

2 Empty nodes in the Penn Treebank

The major types of empty category in the Penn Treebank (PTB) are shown in Table 1, along with their distribution in section 24 of the Wall Street Journal portion of the PTB.

Empty category type	**Count**	**Description**
NP *	1044	NP trace or PRO
NP *T*	265	Trace of WHNP
U	227	Empty unit
0	178	Empty complementizer
ADVP *T*	97	Trace of WHADVP
S *T*	76	Trace of topicalized quoted S
WHNP 0	43	Null WHNP
SBAR	41	Trace of topicalized non-quoted S
WHADVP 0	25	Null WHADVP
others	95	
Total:	**2091**	

Table 1: Common empty categories and their distribution in section 24 of the PTB

A detailed description of the categories and their uses in the treebank is provided in Chapter 4 of the annotation guidelines (Bies et al., 1995). Following Johnson (2002) and Dienes and Dubey (2003a), the compound empty SBAR consisting of an empty complementizer followed by *T* of category S is treated as a single item for purposes of evaluation. This compound category is labeled SBAR in Table 1.

The PTB annotation in general, but especially the annotation of empty categories, follows a modified version of Government-Binding (GB) theory (Chomsky, 1981). In GB, the existence and location of empty categories is determined by the interaction of linguistic principles. In addition, the type of a given empty category is determined by its syntactic context, with the result that the various types of empty category are in complementary distribution. For example, the GB categories NP-trace and PRO (which are conflated to a single category in the PTB) occur only in argument positions in which an overt NP could not occur, namely as the object of a passive verb or as the subject of certain kinds of infinitive.

3 Previous work

Previous approaches to this task have all been learning-based. Collins' (1997) Model 3 integrates the detection and resolution of WH-traces in relative clauses into a lexicalized PCFG. Collins' results are not directly comparable to the works cited below, since he does not provide a separate evaluation of the empty category detection and resolution task.

Johnson (2002) proposes a pattern-matching algorithm, in which the minimal connected tree fragments containing an empty node and its antecedent(s) are extracted from the training corpus, and matched at runtime to an input tree. As in the present approach, Johnson inserts empty nodes as a post-process on an existing tree. He proposes an evaluation metric (discussed further below), and presents results for both detection and detection plus resolution, given two different kinds of input: perfect trees (with empty nodes removed) and parser output.

Dienes and Dubey (2003a,b), on the other hand, integrate their empty node resolution algorithm into their own PCFG parser. They first locate empty nodes in the string, taking a POS-tagged string as input, and outputting a POS-tagged string with labeled empty nodes inserted. The PCFG parser is then trained, using the enhanced strings as input, without inserting any additional empty nodes. Antecedent resolution is handled by a separate post-process. Using Johnson's (2002) evaluation metric, Dienes and Dubey present

results on the detection task alone (i.e., inserting empty categories into the POS-tagged string), as well as on the combined detection and resolution tasks in combination with their parser.[2]

Higgins (2003) considers only the detection and resolution of WH-traces, and only evaluates the results given perfect input. Higgins' method, like Johnson's (2002) and the present one, involves post-processing of trees. Higgins' results are not directly comparable to the other works cited, since he assumes all WH-phrases as given, even those that are themselves empty.

4 The recovery algorithm

4.1 The algorithm

The proposed algorithm for recovering empty categories is shown in Figure 1; the algorithm walks the tree from top to bottom, at each node X deterministically inserting an empty category of a given type (usually as a daughter of X) if the syntactic context for that type is met by X. It makes four separate passes over the tree, on each pass applying a different set of rules.

```
1  for each tree, iterate over nodes from top down
2    for each node X
3      try to insert NP* in X
4      try to insert 0 in X
5      try to insert WHNP 0  or WHADVP 0 in X
6      try to insert *U* in X
7      try to insert a VP ellipsis site in X
8      try to insert S*T* or SBAR in X
9      try to insert trace of topicalized XP in X
10     try to insert trace of extraposition in X
11 for each node X
12     try to insert WH-trace in X
13 for each node X
14     try to insert NP-SBJ * in finite clause X
15 for each node X
16     if X = NP*, try to find antecedent for X
```

Figure 1: Empty category recovery algorithm

The rules called by this algorithm that try to insert empty categories of a particular type specify the syntactic context in which that type of empty category can occur, and if the context exists, specify where to insert the empty category. For example, the category NP*, which conflates the GB categories NP-trace and PRO, occurs typically[3] as the object of a passive verb or as the subject of an infinitive. The rule which tries to insert this category and assign it a function tag is called in line 3 of Figure 1 and given in pseudo-code in Figure 2. Some additional rules are given in the Appendix.

```
if X is a passive VP & X has no complement S
  if there is a postmodifying dangling PP Y
    then insert NP* before all postmodifiers of Y
  else insert NP* before all postmodifiers of X
else if X is a non-finite S and X has no subject
  then insert NP-SBJ* after all premodifiers of X
```

Figure 2: Rule to insert NP*

This rule, which accounts for about half the empty category tokens in the PTB, makes no use of lexical information such as valency of the verb, etc. This is potentially a problem, since in GB the infinitives that can have NP-trace or PRO as subjects (raising and control infinitives) are distinguished from those that can have overt NPs or WH-trace as subjects (exceptional-Case-marked, or ECM, infinitives), and the distinction relies on the class of the governing verb.

Nevertheless, the rules that insert empty nodes do not have access to a lexicon, and very little lexical information is encoded in the rules: reference is made in the rules to individual function words such as complementizers, auxiliaries, and the infinitival marker *to*, but never to lexical properties of content words such as valency or the raising/ECM distinction. In fact, the only reference to content words at all is in the rule which tries to insert null WH-phrases, called in line 5 of Figure 1: when this rule has found a relative clause in which it needs to insert a null WH-phrase, it checks if the head of the NP the relative clause modifies is *reason(s)*, *way(s)*, *time(s)*, *day(s)*, or *place(s)*; if it is, then it inserts WHADVP with the appropriate function tag, rather than WHNP.

The rule shown in Figure 2 depends for its successful application on the system's being able to identify passives, non-finite sentences, heads of phrases (to identify pre- and post-modifiers), and functional information such as subject; similar information is accessed by the other rules used in the algorithm. Simple functions to identify passives, etc. are therefore called by the implemented versions of these rules. Functional information, such as subject, can be gleaned from the function tags in the treebank annotation; the rules make frequent use of a variety of function tags as they occur on various nodes. The output of

[2] It is unclear whether Dienes and Dubey's evaluation of empty category detection is based on actual tags provided by the annotation (perfect input), or on the output of a POS-tagger.

[3] NP* is used in roles that go beyond the GB notions of NP-trace and PRO, including e.g. the subject of imperatives; see below.

Charniak's parser (Charniak, 2000), however, does not include function tags, so in order for the algorithm to work properly on parser output (see Section 5), additional functions were written to approximate the required tags. Presumably, the accuracy of the algorithm on parser output would be enhanced by accurate prior assignment of the tags to all relevant nodes, as in Blaheta and Charniak (2000) (see also Section 5).

Each empty category insertion rule, in addition to inserting an empty node in the tree, also may assign a function tag to the empty node. This is illustrated in Figure 2, where the final line inserts NP* with the function tag SBJ in the case where it is the subject of an infinitive clause.

The rule that inserts WH-trace (called in line 12 in Figure 1) takes a WHXP needing a trace as input, and walks the tree until an appropriate insertion site is found (see Appendix for a fuller description). Since this rule requires a WHXP as input, and that WHXP may itself be an empty category (inserted by an earlier rule), it is handled in a separate pass through the tree.

A separate rule inserts NP* as the subject in sentences which have no overt subject, and which have not had a subject inserted by any of the other rules. Most commonly, these are imperative sentences, but calling this rule in a separate pass through the tree, as in Figure 1, ensures that any subject position missed by the other rules is filled.

Finally, a separate rule tries to find an antecedent for NP* under certain conditions. The antecedent of NP* may be an empty node inserted by rules in any of the first three passes through the tree, even the subject of an imperative; therefore this rule is applied in a separate pass through the tree. This rule is also fairly simple, assigning the local subject as antecedent for a non-subject NP*, while for an NP* in the subject position of a non-finite S it searches up the tree, given certain locality conditions, for another NP subject.

All the rules that insert empty categories are fairly simple, and derive straighforwardly from standard GB theory and from the annotation guidelines. The most complex rule is the rule that inserts WH-trace when it finds a WHXP daughter of SBAR; most are about as simple as the rule shown in Figure 2, some more so. Representative examples are given in the Appendix.

4.2 Development method

After implementing the algorithm, it was run over sections 1, 3, and 11 of the WSJ portion of the PTB, followed by manual inspection of the trees to perform error analysis, with revisions made as necessary to correct errors. Initially sections 22 and 24 were used for development testing. However, it was found that these two sections differ from each other substantially with respect to the annotation of antecedents of NP* (which is described somewhat vaguely in the annotation guidelines), so all of sections 2-21 were used as a development test corpus. Section 23 was used only for the final evaluation, reported in Section 5 below.

5 Evaluation

Following Johnson (2002), the system was evaluated on two different kinds of input: first, on perfect input, i.e., PTB annotations stripped of all empty categories and information related to them; and second, on imperfect input, in this case the output of Charniak's (2000) parser. Each is discussed in turn below.

5.1 Perfect input

The system was run on PTB trees stripped of all empty categories. To facilitate comparison to previous approaches, we used Johnson's label and string position evaluation metric, according to which an empty node is identified by its label plus its string position, and evaluated the detection task alone. We then evaluated detection and resolution combined, identifying each empty category as before, plus the label and string position of its antecedent, if any, again following Johnson's work.

The results are shown in Table 2. Precision here and throughout is the percentage of empty nodes proposed by the system that are in the gold standard (section 23 of the PTB), recall is the percentage of empty nodes in the gold standard that are proposed by the system, and F_1 is balanced f-measure; i.e., 2PR/(P+R).

Task	Prec.	Rec.	F_1
Detection only	94.9	91.1	93.0
Detection + resolution	90.1	86.6	88.4

Table 2: Detection and resolution of empty categories given perfect input (label + string position method), expressed as percentage

These results compare favorably to previously reported results, exceeding them mainly by achieving higher recall. Johnson (2002) reports 93% precision and 83% recall (F_1 = 88%) for the detection task alone, and 80% precision and 70% recall (F_1 = 75%) for detection plus resolution. In contrast to Johnson (2002) and the present work, Dienes and Dubey (2003a) take a POS-tagged string, rather than a tree, as input; they report 86.5% precision and 72.9% recall (F_1 = 79.1%) on the detection task. For Dienes and Dubey, the further task of finding antecedents for empty

categories is integrated with their own PCFG parser, so they report no numbers directly relevant to the task of detection and resolution given perfect input.

5.2 Parser output

The system was also run using as input the output of Charniak's parser (Charniak, 2000). The results, again using the label and string position method, are given in Table 3.

Task	Prec.	Rec.	F_1
Detection only	85.2	81.7	83.4
Detection + resolution	78.3	75.1	76.7

Table 3: Detection and resolution of empty categories on parser output (label + string position method), expressed as percentage

Again the results exceed those previously reported. Johnson (2002) reports 85% precision and 74% recall (F_1 = 79%) for detection and 73% precision and 63% recall (F_1 = 68%) for detection plus resolution on the output of Charniak's parser. Dienes and Dubey (2003b) integrate the results of their detection task into their own PCFG parser, and report 81.5% precision and 68.7% recall (F_1 = 74.6%) on the combined task of detection and resolution.

5.3 Perfect input with no function tags

The lower results on parser output obviously reflect errors introduced by the parser, but may also be due to the parser not outputting function tags on any nodes. As mentioned in Section 4, it is believed that the results of the current method on parser output would improve if that output were reliably assigned function tags, perhaps along the lines of Blaheta and Charniak (2000).

Testing this hypothesis directly is beyond the scope of the present work, but a simple experiment can give some idea of the extent to which the current algorithm relies on function tags in the input. The system was run on PTB trees with all nodes stripped of function tags; the results are given in Table 4.

Task	Prec.	Rec.	F_1
Detection only	94.1	89.5	91.7
Detection + resolution	89.5	85.2	87.3

Table 4: Detection and resolution of empty categories on PTB input without function tags (label + string position method), expressed as percentage

While not as good as the results on perfect input with function tags, these results are much better than the results on parser output. This suggests that function tag assignment should improve the results shown on parser output, but that the greater part of the difference between the results on perfect input and on parser output is due to errors introduced by the parser.

5.4 Refining the evaluation

The results reported in the previous subsections are quite good, and demonstrate that the current approach outperforms previously reported approaches on the detection and resolution of empty categories. In this subsection some refinements to the evaluation method are considered.

The label and string position method is useful if one sees the task as inserting empty nodes into a string, and thus is quite useful for evaluating systems that detect empty categories without parse trees, as in Dienes and Dubey (2003a). However, if the task is to insert empty nodes into a tree, then the method leads both to false positives and to false negatives. Suppose for example that the sentence *When do you expect to finish?* has the bracketing shown below, where '1' and '2' indicate two possible locations in the tree for the trace of the WHADVP:

When do you [$_{VP}$ expect to [$_{VP}$ finish 1] 2]

Suppose position 1 is correct; i.e. it represents the position of the trace in the gold standard. Since 1 and 2 correspond to the same string position, if a system inserts the trace in position 2, the string position evaluation method will count it as correct.

This is a serious problem with the string-based method of evaluation, if one assumes, as seems reasonable, that the purpose of inserting empty categories into trees is to be able to recover semantic information such as predicate-argument structure and modification relations. In the above example, it is clearly semantically relevant whether the system proposes that *when* modifies *expect* instead of *finish*.

Conversely, suppose the sentence *Who (besides me) cares?* has the bracketing shown:

Who [$_{S}$ 1 (besides me) 2 [$_{VP}$ cares]]

Again suppose that position 1 represents the placement of the WHNP trace in the gold standard. If a system places the trace in position 2 instead, the string position method will count it as an error, since 1 and 2 have different string positions. However it is not at all clear what it means to say that one of those two positions is correct and the other not, since there is no semantic, grammatical, or textual indicator of its exact position. If the task

is to be able to recover semantic information using traces, then it does not matter in this case whether the system inserts the trace to the left or to the right of the parenthetical.

Given that both false positives and false negatives are possible, I propose that future evaluations of this task should identify empty categories by their label and by their parent category, instead of, or perhaps in addition to, doing so by label and string position. Since the parent of an empty node is always an overt node[4], the parent could be identified by its label and string position (left and right edges). Resolution is evaluated by a natural extension, by identifying the antecedent (which could itself be an empty category) according to its label and its parent's label and string position. This would serve to identify an empty category by its position in the tree, rather than in the string, and would avoid the false positives and false negatives described above.

In addition to an evaluation based on tree position rather than string position, I propose to evaluate the entire recovery task, i.e., including function tag assignment, not just detection and resolution.

The revised evaluation is still not perfect: when inserting an NP* or NP*T* into a double-object construction, it clearly matters semantically whether it is the first or second object, though both positions have the same parent.[5] Ideally, we would evaluate based on a richer set of grammatical relations than are annotated in the PTB, or perhaps based on thematic roles. However, it is difficult to see how to accomplish this without additional annotation. It is probable that constructions of this sort are relatively rare in the PTB in any case, so for now the proposed evaluation method, however imperfect, will suffice.

The result of this revised evaluation, given perfect input, is presented in Table 5. The first two rows are comparable to the string-based results in Table 2; the last row, showing the results of the full recovery task (i.e., including antecedents and function tags), is not much lower, suggesting that labeling empty categories with function tags does not pose any serious difficulties.

[4] The only exception is the 0 complementizer and S*T* daughters of the SBAR category in Table 1; but since the entire SBAR is treated as a single empty node for evaluation purposes, this does not pose a problem.

[5] I am indebted to two ACL reviewers for calling this to my attention.

Task	Prec.	Rec.	F_1
Detection only	95.6	91.9	93.7
Detection + resolution	90.8	87.3	89.0
Recovery (det.+res.+func. tags)	89.8	86.3	88.0

Table 5: Detection, resolution and recovery of empty categories given perfect input (label + parent method), expressed as percentage

Three similar evaluations were also run, using parser output as input to the algorithm; the results are given in Table 6.

Task	Prec.	Rec.	F_1
Detection only	78.4	75.2	76.7
Detection + resolution	72.3	69.3	70.8
Recovery (det.+res.+func. tags)	69.7	66.8	68.2

Table 6: Detection, resolution and recovery of empty categories on parser output (label + parent method), expressed as percentage

The results here are less impressive, no doubt reflecting errors introduced by the parser in the labeling and bracketing of the parent category, a problem which does not affect a string-based evaluation. However it does not seem reasonable to have an effective evaluation of empty node insertion in parser output that does not depend to some extent on the correctness of the parse. The fact that our proposed evaluation metric depends more heavily on the accuracy of the input structure may be an unavoidable consequence of using a tree-based evaluation.

6 Discussion

The empty category recovery algorithm reported on here outperforms previously published approaches on the detection and resolution tasks; it also does well on the task of function tag assignment to empty categories, which has not been considered in other work. As suggested in the introduction, the reason a rule-based approach works so well in this domain may be that empty categories are not naturally in the text, but are only inserted by the annotator, who is consciously following explicit linguistic principles, in this case, the principles of early GB theory.

As a result, the recovery of empty categories is, for the most part, more amenable to a rule-based approach than to a learning approach. It makes little sense to learn, for example, that NP* occurs as the object of a passive verb or as the subject of certain infinitives in the PTB, if that information is already explicit in the annotation guidelines.

This is not to say that learning approaches have nothing to contribute to this task. Information

about individual lexical items, such as valency, the raising/ECM distinction, or subject vs. object control, which is presumably most robustly acquired from large amounts of data, would probably help in the task of detecting certain empty categories.

Consider for example an input structure *V [$_S$ to VP]*. GB principles, which are enforced in the annotation guidelines, dictate that an empty category must be inserted as the subject of the infinitival S; but exactly which empty category, NP* or NP*T*, depends on properties of the governing verb, including whether it is a raising or control verb, such as *seem* or *try*, or an ECM verb, such as *believe*. In the present algorithm, the rule that inserts NP* applies first, without access to lexical information of any kind, so NP* is inserted, instead of NP*T*, regardless of the value of V. This leads to some errors which might be corrected given learned lexical information. Such errors are fewer than might have been expected, however: the present system achieved 97.7% precision and 97.3% recall (F_1 = 97.5%) on the isolated task of detecting NP*, even without lexical knowledge (see Table 7).

A combined learning and rule-based algorithm might stand to make a bigger gain in the task of deciding whether NP* in subject position has an antecedent or not, and if it does, whether the antecedent is a subject or not. The annotation guidelines and the theory that underlies it are less explicit on the principles underlying this task than they are on the other subtasks. As a result, the accuracy of the current system drops considerably when this task is taken into account, from 97.5% F_1 to 86.9% (see Table 7). Dienes and Dubey (2003a), on the other hand, claim this as one of the strengths of their learning-based system.

Empty category type	**Detection only (F_1)**	**Detection + resolution (F_1)**
NP*	97.5	86.9
NP*T*	96.2	96.0
U	98.6	-
0	98.5	-
ADVP*T*	79.9	79.9
S*T*	92.7	92.7
WHNP 0	92.4	-
SBAR	84.4	84.4
WHADVP 0	73.3	-

Table 7: F_1 for detection and resolution of empty categories by type, using perfect input (label + parent method), expressed as percentage

7 Conclusion

In this paper I have presented an algorithm for the recovery of empty categories in PTB-style trees that otherwise lack them. Unlike previous approaches, the current algorithm is rule-based rather than learning-based, which I have argued is appropriate for this task, given the highly theoretical nature of empty categories in the PTB. Moreover, the algorithm has no access to lexical information such as valency or verb class.

Using the string-based evaluation metric proposed by Johnson (2002), the current system outperforms previously published algorithms on detection alone, as well as on detection combined with resolution, both on perfect input and in combination with parsing. In addition, we have performed additional evaluation using a tree-based metric, and including an evaluation of function tag assignment as well.

8 Acknowledgements

I would like to thank Simon Corston-Oliver, Mark Johnson, and Hisami Suzuki for their helpful input.

References

Bies, A., M. Ferguson, K. Katz and R. MacIntyre. 1995. *Bracketing Guidelines for Treebank II style Penn Treebank Project*. Linguistic Data Consortium.

Blaheta, D. and E. Charniak. 2000. Assigning Function Tags to Parsed Text. In *Proceedings of the North American Chapter of the Association for Computational Linguistics*, pages 234-240.

Charniak, E. 2000. A maximum-entropy-inspired parser. In In *Proceedings of the North American Chapter of the Association for Computational Linguistics*, pages 132-139.

Chomsky, N. 1981. *Lectures on Government and Binding*. Foris Publications, Dordrecht.

Collins, M. 1997. Three Generative, Lexicalised Models for Statistical Parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pages 16-23.

Dienes, P. and A. Dubey. 2003a. Deep Syntactic Processing by Combining Shallow Methods. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 431-438.

Dienes, P. and A. Dubey. 2003b. Antecedent Recovery: Experiments with a Trace Tagger. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 33-40.

Higgins, D. 2003. A machine-learning approach to the identification of WH gaps. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics*, pages 99-102.

Johnson, M. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 136-143.

Marcus, M., B. Santorini and M.A.Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313-330.

Appendix: Sample rules

To insert 0 Comp:
if X=SBAR & !Comp(X) & !WHXP daughter(X)
& ∃ S daughter Y of X
& !(parent(X)=NP & sister(X)=NP)
    then insert 0 to left of Y

To insert WHNP/WHADVP:
if X=SBAR & parent(X)=NP
& sister(X)=NP & !Comp(X)
& !WHXP daughter(X) & ∃ S daughter Y of X
    if head(parent(X)) in {reason(s) way(s)
    time(s) day(s) place(s)}
        then insert WHADVP to left of Y
    else insert WHNP to left of Y

To insert *U*:
insert *U* / $ CD+ _

To insert WH-trace:
if X=SBAR & ∃ S daughter Y of X
& ∃ WHXP daughter W of X
    then find trace(W) in Y

To find trace(W) in X:

insert trace:
(for W = WHXP, insert XP*T*)
if X has conjuncts
    then find trace(W) in each conjunct of X
else if X has a PP daughter Y with no object
& W=WHNP
    then insert *T* to right of P
else if X=S and !subject(X) & W=WHNP
    then insert *T* as last pre-mod of X
else if X contains a VP Y
    then find trace(W) in Y
else if X contains ADJP or clausal complement Y
        & W=WHNP
    then find trace(W) in Y
else if W=WHNP
& ∃ infinival rel. clause R, R=sister(W)
& X=VP & X has an object NP
& subject(R) is an empty node E
    then insert *T* as last pre-mod of R
    then delete E
else if W=WHNP
    then insert *T* as first post-mod of X
else insert *T* as last post-mod of X

assign function tag:
if W = WHNP & *T* a pre-mod of S
    then assign 'SBJ' to *T*
if W = WHADVP & W is not empty
    if W = 'why'
        then assign 'PRP' to *T*
    if W = 'when'
        then assign 'TMP' to *T*
    if W = 'where'
        then assign 'LOC' to *T*
    if W = 'how'
        then assign 'MNR' to *T*
else if W = WHADVP & parent(parent(W)) =NP
    if head(sister(parent(W))) = 'reason(s)'
        then assign 'PRP' to *T*
    if head(sister(parent(W)))='time(s)' or 'day(s)'
        then assign 'TMP' to *T*
    if head(sister(parent(W))) = 'place(s)'
        then assign 'LOC' to *T*
    if head(sister(parent(W))) = 'way(s)'
        then assign 'MNR' to *T*