Semi-Automatic Generation of F-Structures from Treebanks

Josef van Genabith
Dublin City University
josef@compapp.dcu.ie

Andy Way
Dublin City University
away@compap.dcu.ie

Louisa Sadler
University of Essex
louisa@essex.ac.uk

Proceedings of the LFG99 Conference

The University of Manchester

Miriam Butt and Tracy Holloway King (Editors)

1999

CSLI Publications

http://www-csli.stanford.edu/publications/

Abstract

Statistical approaches to processing Lexical Functional Grammars (LFG-DOP, [1]) require large corpora of text annotated with c-structure and f-structure representations. To date, such corpora that exist are constructed manually or semi-automatically. Manual construction is both time-consuming and error-prone. Semi-automatic construction usually proceeds as follows: an existing LFG grammar is used to parse input text. Typically, for each sentence in the input text, parsing will produce a large number of c- and f-structure analyses. A linguistic expert then inspects the analyses and for each sentence in the input text selects the single best analysis for the case at hand. For large grammars this can involve inspection of hundreds or thousands of proposed analyses for a single input sentence. In the present paper we develop an alternative, semi-automatic methodology that as much as possible avoids manual inspection of analyses for best fit. As input, the method requires a treebank, from which the corresponding CF-PSG is automatically compiled following the method of ([3]). This CF-PSG is then manually annotated with functional information, after which the treebank representations (not the strings) are "reparsed", simply following the c-structure provided by the original annotators, thereby inducing f-structures corresponding to the original c-structures. If the f-structure equations encountered during the "reparse" are deterministic, then the whole process is. In addition to producing a treebank with f-structure annotations, our method can be used to produce a stand-alone LFG grammar from the treebank resource. In order to do this we need proper semantic forms to implement the LFG treatment of subcategorization. We show how semantic forms can be compiled automatically given the f-structure resources produced by our original method. Finally, given that the c-structure backbone automatically compiled from the original treebank does not bear much resemblance to c-structure components designed manually by linguists, we develop a simple structure-preserving grammar compaction technique which brings the compiled c-structure backbone more in line with standard linguistic practice. The paper updates and extends earlier reports on our research in ([17, 18]). The resources obtained are available at the following webpage: `http://www.compapp.dcu.ie/~away/Treebank/treebank.html`

1 Introduction

In order to ensure good performance, statistical approaches to natural language processing require good quality training corpora. If the corpus quality is poor, then so is the quality of the statistical methods trained on them. In addition, statistical methods require substantial training corpora, particularly so if, as is the case with LFG-DOP ([1]), the methods involve a large number of distinct observable events. Owing to these quality and size requirements, construction of suitable corpora tends to be both time-consuming and expensive.

To date, training corpora are constructed manually or semi-automatically. In manual corpus construction, human experts annotate text, which is time-consuming, expensive and error-prone. Expert annotators are hard to find and it is often difficult to ensure consistency between different annotators.

Semi-automatic corpus construction usually proceeds as follows: an existing grammar[1] is used to parse the input text. For a large coverage grammar this will typically yield a substantial num-

[1] In real life this often means *developing* a grammar ... (sometimes from scratch).

ber of analyses for each input sentence caused by lexical and structural ambiguities such as PP-attachment, scope of conjunctions and other operators, and part of speech ambiguities. A highly trained human expert then has to select the best analysis for inclusion in the training corpus. In the case of realistic LFG fragments, this can easily amount to sifting through hundreds or thousands of c- and f-structure pairings for a single input sentence. Again this is time-consuming, error-prone and requires extensive linguistic expertise. A number of ranking methods have been proposed to address this problem, including the use of optimality ([5]) or statistical filters ([14]). In such approaches, given some threshold, a human expert selects from the highest scoring analyses provided by the system. However, none of the approaches proposed so far guarantees to find the best analysis among the highest ranked proposals, given the input data. This means that in order to achieve the best quality for the training corpus, the human expert has to sift through all analyses or manually repair highly ranked analyses.

In the present paper we develop an alternative methodology, which whilst still semi-automatic in nature, as far as possible avoids the necessity of laborious scrutiny of a number of candidate solutions. As input our method requires a treebank (i.e. a *parsed*–rather than merely tagged–corpus of sentences). A number of such resources are available, including: the Penn Treebank ([15]), the Lancaster Parsed Corpus ([6]), the SUSANNE Corpus ([16]), the TOSCA Corpus ([8]), the Lancaster-Leeds Treebank ([16], pp.16–19), the Air Travel Information System (ATIS) Corpus ([9], [15]), and the Associated Press (AP) Treebank ([13]), a skeleton-parsed corpus of American newswire reports, approximately 1,000,000 words in length. This corpus is of particular interest as it consists of short, complete texts on a number of subjects. For the experiments reported in this paper used the publicly available sub-corpus of the first 100 sentences of the AP newswire.

From the treebank fragment we automatically induce the corresponding CF-PSG, following the method developed in ([3]). We then manually annotate the resulting CF-PSG with functional schemata and provide macros associating lexical tags with f-structures. We then "reparse" - and this is the trick - the treebank representations (not the strings), simply following the c-structure provided by the original annotations, thereby inducing f-structures corresponding to the original c-structures. The potential advantages of our semi-automatic method are twofold:

1. We do not end up with a multitude of c-structure parses due to lexical and structural ambiguity. The reason is that our "reparsing" simply follows the *single* structural analysis provided by the original annotator. This analysis is deemed to provide the best fit given the data. If the f-structure equations encountered (and resolved) during this process are deterministic, then the whole process is and we come up with a *single* f-structure.

2. Rule annotations cumulatively feed into the construction of an LFG grammar, which is a valuable resource in itself. Thus providing these annotations is a task of a more general nature than manually selecting suggested analyses from a parse. Annotated rules are generative, and so potentially apply to cases not yet seen.

The paper presents four experiments. In the first we detail the approach taken, describe an initial experiment and discuss some advantages and disadvantages of the method proposed. The first experiment involves a "deterministic" (deterministic on reparsing treebank entries) grammar, G1, and produces basic f-structures where `pred` values are mostly surface word forms. The grammar is not yet a stand-alone LFG and its CF-PSG backbone is rather large. In addition to providing feature structure annotations to treebanks, our method can also be used to compile other linguistic resources. In order to obtain an stand-alone LFG resource, we need to implement the

LFG account of subcategorization in terms of semantic forms, a distinction between subcategorizable and non-subcategorizable grammatical functions and the global f-structure completeness and coherence well-formedness constraints ([10]). In a second experiment we show how, given the feature structure annotations provided by our original approach (experiment one, G1), semantic forms can be compiled automatically, effectively extending the CF-PSG compilation method of ([3]) to higher level linguistic representations. The deterministic approach documented in experiment one does not always get things right. An area which is particularly delicate in this respect is the distinction between adjuncts and subcategorized complements. In order to improve the quality of the f-structures and hence the quality of the semantic forms computed from those f-structures, in experiment three we introduce a very limited amount of disjunction in the f-structure annotations resulting in a slightly non-deterministic grammar G2. We also recompute semantic forms from the feature structure annotations induced by G2 to obtain an improved stand-alone LFG. Experiment four focuses on the "quality" of the CF-PSG backbone obtained from the original treebank using the compilation method of ([3]). This CF-PSG is very different from LFG c-structure components designed manually by linguists in that it features a very large number of highly specific and often only minimally different rules. We develop a simple grammar compaction technique which brings the compiled c-structure backbone more in line with linguistic practice. This experiment results in a "collapsed", non-deterministic grammar G3. In contrast to the method presented in ([11]), our technique is structure preserving and guaranteed to respect linguistically motivated structure. The paper updates and extends earlier reports on our research in ([17, 18]). The resources and results generated in the four experiments are available for inspection at `http://www.compapp.dcu.ie/~away/Treebank/treebank.html`

Finally, the point of our exercise is not so much to produce the perfect LFG resource (which our method cannot, naturally), but rather to present and explore a methodology which, given a tree bank as a starting point, shows promise.

2 From Treebanks to F-Structures: Experiment 1

2.1 The AP Treebank

As input data we use the publicly available subset (first 100 $) of the AP treebank developed at the University of Lancaster ([13]). The full corpus contains over 100,000 sentences drawn from American newswire reports. The corpus is tagged, post-edited and hand parsed. The annotation schema employs 183 lexical tags and 53 non-terminal categories. The shortest sentence in the treebank fragment has 4 words, the longest 43 words. As an example, consider

(1)
```
A001   1  v
[N[G Alvis_NP1 Rogers_NP1 '_$ G] 18-foot_NNU1 [ jump_NN1 shot_NN1 ][
with_IW [N one_MC1 second_NNT1 N][Tg remaining_VVG Tg]]N][V gave_VVD
[N Wake_NP1 Forest_NNL1 N][N a_AT1 79-77_MC-MC upset_JJ victory_NN1 [P
over_II [N 13th-ranked_JJ Virginia_NP1 N]P]N] Saturday_NPD1 [P in_II
[N an_AT1 [ Atlantic_NP1 Coast_NNL1 Conference_NNJ1 ] contest_NN1 N]
P]V] ._.
```

The corpus entry data structure has 4 slots. The first two slots are source article and sentence identifiers. The third slot indicates whether the sentence which follows in the fourth slot is grammatical

(v, as here) or not. Word class information is attached to the word form as in `word_TAG`. Tree structure is encoded in terms of labelled bracketing. Punctuation markers are coded as punctuation symbol followed by an underscore followed by punctuation symbol. Since all corpus entries are sentences, this information is not explicitly coded in the representations. The corpus is 'skeletally parsed', where some partial parsing is permitted ([12], pp.36–37):

> "The tree is incomplete: in some cases the brackets are left unlabelled, and this was allowed in order to give annotators a chance to decide that some sequence of words was a constituent, without committing themselves to a label. The simplicity of the scheme was intended not only to speed up the process of treebank compilation, but also to limit the intellectual complexity of the task of human parsing, so that inconsistencies and inaccuracies in treebanking practice were minimised."

In (1) above, both

`[ jump_NN1 shot_NN1 ]` and `[with_IW [N one_MC1 second_NNT1 N][Tg remaining_VVG Tg]]`

have not been allocated a mother category. In a pre-editing step, unlabelled brackets are removed from the input tree-bank. Thus slightly flatter representations are produced. This is the only pre-editing step in our approach.

2.2 Extracting the CF-PSG

To illustrate our method we will consider the following simple example entry from the AP corpus:[2]

(2)
```
A001  39 v
[N The_AT march_NN1 N][V was_VBDZ [J peaceful_JJ J]V] ._.
```

Our system is implemented in Prolog. In order to extract the CF-PSG from the corpus entries, we first feed the corpus into a pre-compiler that renders the corpus entries Prolog compatible. This step includes upper to lower case and labelled bracketing to Prolog term conversion. The output of the compilation step on our example sentence (2) is:

(3)
```
tree(a001,39,v,s(n(at(the),nn1(march)),v(vbdz(was),j(jj(peaceful)))))
```

We then feed the pre-processed corpus into a CF-PSG extraction module, which, following the method developed in ([3]), reads off the CF-PSG rules by recursively traversing local sub-trees in the corpus entry. Lexical entries are simply non-branching terminal trees of depth 2. Given our example sentence (2), the CF-PSG extraction module yields:

(4)
```
lex(at(the)).          rule(n(A), [at(B),nn1(C)]).
lex(nn1(march)).       rule(j(A), [jj(B)]).
lex(vbdz(was)).        rule(v(A), [vbdz(B),j(C)]).
lex(jj(peaceful)).     rule(s(A), [n(B),v(C)]).
```

Note that the CF-PSG extraction module adds a unique Prolog variable to each category in the grammar rules. In the annotated grammar, variables associate tree nodes with f-structure components.

[2] For lack of space we will only be able to present very simple examples in the paper.

T he treebank fragment contains:

non-lexical subtrees	> 1300	words	> 2000
n-type phrases	> 500	p-type phrases	> 200
v-type phrases	> 200	s-type phrases	> 100

From this we compile:

grammar rules	> 500	lexical entries	> 700

The extracted grammar is available in a number of versions: as a simple CF-PSG, as a probabilistic CF-PSG, as a Prolog DCG and in a simple chart parsing framework.

2.3 Support Tools

The annotation and debugging tasks are supported by a number of tools:

- We compiled a version of the CF-PSG which indicates where each rule and lexical entry is compiled from. This information aids the grammar annotator in that s/he can quickly inspect the sentences which give rise to particular grammar rules and thus adjust annotations accordingly.
- We constructed a tool that given a treebank entry and an annotated grammar gives all possible partial reparsing analyses. This is especially useful to locate bugs if a particular treebank entry cannot yet be completely reparsed by a grammar whose annotations are still under development.
- We compile semantic forms from the f-structures generated (see Section 3 below) and an indexed list which f-structures particular semantic forms are derived from. Inspection of this list identifies odd-looking semantic forms and hence f-structures.

2.4 Annotating the Grammar

Lexical Functional Grammars ([10]) are decorated grammars with a CF-PSG backbone. A simple LFG rule fragment for our example sentence (2) would look as follows:

(5)
$$\begin{array}{lcl} \text{S} \rightarrow & \overset{\text{N}}{(\uparrow \text{SUBJ}) = \downarrow} & \overset{\text{V}}{\uparrow = \downarrow} \\ \text{V} \rightarrow & \overset{\text{VDBZ}}{\uparrow = \downarrow} & \overset{\text{J}}{(\uparrow \text{XCOMP}) = \downarrow} \end{array} \qquad \begin{array}{lcl} \text{N} \rightarrow & \overset{\text{AT}}{\uparrow = \downarrow} & \overset{\text{NN1}}{\uparrow = \downarrow} \\ \text{J} \rightarrow & \overset{\text{JJ}}{\uparrow = \downarrow} & \end{array}$$

The decorations are referred to as functional schemata. They are statements in an equality logic describing f-structures and, given a parse tree, define a correspondence between f-structure and nodes in the parse tree.

The corresponding lexical entries are:

(6)

AT *the*
$(\uparrow \text{SPEC}) = \text{the}$

N *march*
$(\uparrow \text{PRED}) = \text{march}$
$(\uparrow \text{NUM}) = \text{SG}$

VDBZ *was*
$(\uparrow \text{PRED}) = \text{be}\langle\uparrow \text{SUBJ}, \uparrow \text{XCOMP}\rangle$
$(\uparrow \text{SUBJ NUM}) = \text{SG}$
$(\uparrow \text{TENSE}) = \text{PAST}$

JJ *peaceful*
$(\uparrow \text{PRED}) = \text{peaceful}$

Together these map our example string (2) into the following f-structure:[3]

(7)

$$\begin{bmatrix} \text{SUBJ} & \begin{bmatrix} \text{PRED} & \text{march} \\ \text{SPEC} & \text{the} \\ \text{NUM} & \text{SG} \end{bmatrix} \\ \text{PRED} & \text{be}\langle\uparrow \text{SUBJ}, \uparrow \text{XCOMP}\rangle \\ \text{TENSE} & \text{PAST} \\ \text{XCOMP} & \begin{bmatrix} \text{PRED} & \text{peaceful} \end{bmatrix} \end{bmatrix}$$

We model this LFG grammar by using a simple version of graph unification described in ([7]). This version provides a subset of the constraint language of LFG. Essentially it supports positive constraints over finite paths, but it does not provide negation, regular path expressions or constraining '$=_c$' equations. In order to capture the full constraint language provided by LFG, more sophisticated constraint resolution mechanisms would have to be used, such as those provided by the XLE (The Xerox Linguistic Environment).

In our approach we associate lexical categories in the AP tag set with macros. Each macro defines an f-structure fragment for the set of words in the tag class. Those corresponding to the lexical entries in (6) are:

(8)

```
macro(at(Word),FStr) :-
  FStr:spec === Word.

macro(nn1(Word),FStr) :-
  FStr:pred === Word,
  FStr:num  === sg.
```

```
macro(jj(Word),FStr) :-
  FStr:pred === Word.

macro(vbdz(_Word),FStr) :-
  FStr:tense  === past,
  FStr:pred   === be.
```

Note that for simplicity we often use word forms as `pred` values. A more sophisticated approach would employ a morphological analysis component to extract root forms.

[3]Note that this particular f-structure is incomplete in that the adjectival complement *peaceful* shares a subject with the verb *be*.

Our rule annotations are direct translations of the LFG rules given in (5) into the simple graph unification framework:

(9)
```
rule(n(A), [at(B),nn1(C)]) :-
   A === B,
   A === C.

rule(j(A), [jj(B)]) :-
   A === B.

rule(v(A), [vbdz(B),j(C)]) :-
   A === B,
   A:xcomp === C.

rule(sent(A), [n(B),v(C)]) :-
   A:subj === B,
   A      === C.
```

2.5 "Reparsing" the Treebank

Now we have everything in place to "reparse" the Prolog compatible version of the original treebank. Our "reparsing" interpreter reparses the annotated treebank entries simply by deterministically following the tree annotations provided by the original annotators. In so doing the interpreter solves the constraint equations associated with the grammar rules and lexical macros involved in the parse. If the equations are deterministic (i.e. do not contain disjunctions) then the whole process is deterministic and returns a single f-structure which is induced by the single best c-structure analysis (namely the one provided by the original human annotators) of the string in question. For our example sentence the output of the "reparsing" interpreter yields:

(10)
```
subj : spec : the
       pred : march
       num  : sg
xcomp : pred : peaceful
tense : past
pred  : be
```

2.6 Lexical Tags and Macros

The tag set (both lexical and non-lexical) employed in the annotations in the AP corpus is extremely fine-grained. Altogether, the number of lexical categories is 183. This allows us to construct quite specific lexical macros inducing f-structure fragments. To give a example, the total number of noun-like parts of speech (including pronominals) distinguished by the tag set is 58. Consider the following simple section from the noun tag set:

NN	common noun, neutral for number (`sheep, cod, headquarters`)
NN1	singular common noun (`book, girl`)
NN2	plural common noun (`books, girls`)

These tags are associated with the following lexical macros:

(11)
```
macro(nn(Word),FStr) :-
  FStr:pred === Word.

macro(nn1(Word),FStr) :-
  FStr:pred === Word,
  FStr:num  === sg.

macro(nn2(Word),FStr) :-
  FStr:pred === Word,
  FStr:num  === pl.
```

However, there are particular linguistic constructions not covered by the AP tagset: control verbs are a case in point. Since these are not identified by a particular tag, our lexical macros will not associate them with the required functional control equations and the corresponding reentrancies will not show up in the f-structures produced with those macros.

2.7 Rule Specificity

The number total of categories used in the AP tag set is 236 = 183 lexical tags + 53 non-lexical tags. As a consequence, this entails a great variety of minimally different CF-PSG rules extracted from the original annotations. As a rough measure of *rule specificity*, consider the ratio RS between the number R of different rules (types) extracted given the number S of (non-lexical) local subtrees (tokens) in the corpus:

$$RS := \frac{R}{S}$$

We have that $1 \geq RS > 0$. If each local subtree in the corpus is defined by a different rule we have that $R = S$ and $RS = 1$. On the other hand, if each local subtree in the corpus is predicted by one and the same rule we have that $R = 1$ and $RS = 1/S$ which for large S approximates to 0. Thus, a high rule specificity value indicates that the rules relative to the corpus fragment considered are very specific: they do not seem to express significant generalizations and apply to very few cases. A low rule specificity value indicates that rules apply to many cases, hence they are likely to express generalizations. Notice that this definition of rule specificity is not an absolute measure but relative to a particular corpus. Consider the following pathological case: one might be presented with a corpus fragment C_1 where each local subtree is defined by a different rule. This would result in a RS_{C_1} measure of 1. Now if one were to increase the size of the corpus fragment $C_2 \supset C_1$ one may find oneself in the situation that all new trees are predicted by the rules already compiled from C_1. Hence $RS_{C_2} < RS_{C_1}$. Theoretically, this situation could continue for larger and larger fragments $C_n \supset \ldots \supset C_2 \supset C_1$. In the limit this means that if one has found a rule set (however large but finite) that defines an infinite number of subtrees in some infinite corpus, the rule specificity of that rule set goes to 0. What is the upshot of this? In comparing two rule sets R_1 and R_2 for the same corpus fragment C the statement that R_1 is more specific than R_2 measured in terms of rule specificity defined above is always relative to C. In general, the larger the size of C the more general the claim that R_1 is more specific than R_2. An absolute measure can be achieved as follows: if the growth rate (that is the first derivative of the function that maps the number of strings in the corpus to the number of rules required to predict the strings) of one of the rule sets R_1 and R_2 goes to 0 (intuitively: no more new rules are required to account for new corpus entries) and if the number of rules in that rule set - say R_1 - is smaller then the number of rules in R_2, then R_1 is less specific than R_2. The cut off point here is the minimal corpus size n where both the

growth rate of R_1 is 0 and $|R_1| < |R_2|$.[4] An alternative but fully equivalent way of looking at this is the following: given a corpus we could define *rule specificity* RS as the ratio between the average number $RT = R/T$ of rule types that go into the construction of a tree and the average number $ST = S/T$ of local subtree tokens in a tree (where T is the number of trees in the corpus). But then we have that $RS = RT/ST = (R/T)/(S/T) = R/S$. Taking the inverse of *rule specificity* we get the average number of times SR a rule applies in the tree bank fragment:

$$SR = RS^{-1} = \frac{S}{R}$$

For our treebank fragment we get the following results: the fragment contains 100 trees and about 1300 local subtrees. From these we extract about 500 rules. This means that on average each tree has about 13 subtrees and again, on average about 5 different rules types contribute to the construction of a single tree. With that we get a rule specificity measure of $RS = 5/13 = 0.38$ and $SR = 2.6$, that is, on average each rule compiled from the treebank fragment applies only 2.6 times in the fragment. In summary:

RS	0.38	SR	2.6
RT	5	ST	13

This is a very high rule specificity measure. The point is illustrated by the following fragment from the set of induced rules:

(12)
```
rule(n(_), [at(_),nn1(_),fn(_)]).
rule(n(_), [at(_),nn1(_),fr(_)]).
rule(n(_), [at(_),nn1(_),nn1(_)]).
rule(n(_), [at(_),nn1(_),p(_)]).
rule(n(_), [at(_),nn1(_),pnct(_),fr(_)]).
rule(n(_), [at(_),nn1(_),tg(_)]).
rule(n(_), [at(_),nn1(_)]).
rule(n(_), [at(_),nn2(_)]).
rule(n(_), [at(_),nnj(_),nn2(_)]).
rule(n(_), [at(_),nnj(_)]).
rule(n(_), [at(_),nnj1(_)]).
rule(n(_), [at(_),nnl1(_),p(_)]).
rule(n(_), [at(_),nnl1(_)]).
rule(n(_), [at(_),nnt1(_)]).
rule(n(_), [at(_),np1(_),jb(_),nn2(_),fr(_)]).
rule(n(_), [at(_),np1(_),jj(_),vvz(_),nn1(_)]).
rule(n(_), [at(_),np1(_),nn1(_),p(_)]).
rule(n(_), [at(_),np1(_),nn1(_)]).
rule(n(_), [at(_),np1(_),nn2(_),pnct(_),nn2(_),cc(_),nn2(_)]).
rule(n(_), [at(_),np1(_),nn2(_)]).
rule(n(_), [at(_),np1(_),np1(_)]).
```

These rules define nominal constructions starting with an article unspecified for number (either *the* or *no* here) followed by a nominal element (and possibly more material). The RHS's of the grammar

[4] It might be the case that the growth rate of R_1 and R_2 does not go to 0. In that case things are more complicated ..

rules have substantial left common prefixes. This is one source of redundancy. The AP tag set is extremely fine-grained: e.g. it discriminates nominal elements into singular common nouns (NN1), genitive singular common nouns (NN1$), plural common nouns (NN2), organisation noun neutral for number (NNJ), singular organisation noun (NNJ1), plural organisation noun (NNJ2), locative noun neutral for number (NNL), singular locative noun (NNL1), plural locative noun (NNL2), numeral noun neutral for number (NNO), singular numeral noun (NNO1), singular proper noun (NP1) and so on. Distinctions such as these contribute significantly to the large number of specific rules which often differ only minimally. A large number of rules have exclusively (or a high count of) lexical categories (tags) in their RHS's. Furthermore, and perhaps surprisingly, there is only very limited recursion in the extracted grammar rules. In fact, less then 5% of our rule fragment are directly recursive rules. Finally, there do not seem to be any global X-bar principles structuring the design of the extracted grammar.

2.8 Annotation Workload

The high rule specificity is both good news and bad news. The fact that many of the rules apply to only very few cases means that we can provide rules with very specific f-structure annotations. This is the reason why the results (i.e. the f-structures) obtained with the deterministic grammar G1 are so surprisingly good. For a more detailed discussion of the quality of the f-structures produced see Section 4 **Evaluation of Experiments 1 and 2** below.

On the other hand the fact that we compile a large CF-PSGs with highly specific but often minimally different individual rules means that the manual rule-annotation part of our method is faced with large numbers of often repetitive rules.

Based on our annotation experience, we estimate rule annotation workload as follows: for each rule we assign a 5 minute time unit. This includes annotation, testing and a revision cycle. For a grammar of 500 rules this results in $500 \times 5\ mins = 2500\ mins \approx 42\ hrs$, i.e. a "good" working week. Our initial annotation time was about half a working week. This was the time required to get the first grammar up and running. Since then the grammar has, of course, undergone a number of further revision and improvement cycles. Of course, annotation time varies greatly with linguistic expertise, and in particular familiarity with the corpus, the tag set and LFG. To a certain extent, the large number of rules extracted from the AP corpus is counterbalanced by the repetitive structure of many of the rules. "Reparsing" in testing and revision cycles tends to be very short (for G1 it is deterministic; it simply follows the original c-structure annotations and solves the f-description constraints along the way).

3 Towards a Stand Alone LFG Resource: Experiment 2

In the preceding section we introduced a methodology which provides semi-automatic LFG annotations for treebanks. The approach is deterministic and, in contrast to alternative approaches, avoids manual inspection of candidate analyses for inclusion in a corpus. The quality of the resulting f-structures is surprisingly good. However, the annotated grammar developed is not a stand-alone LFG resource in that it cannot yet be employed in parsing free strings (in contrast to "reparsing" treebank entries). What is missing is the LFG account of subcategorization in terms of a distinction between subcategorizable and non-subcategorizable grammatical functions, semantic forms and global completeness and coherence well-formedness constraints on f-structures ([10]).

In this section we show how given the resources produced by the methodology described in experiment 1, semantic forms can be compiled automatically. These semantic forms can be folded automatically into our grammars and together with an implementation of coherence and completeness constraints we provide a method for obtaining a stand-alone LFG resource.

3.1 Semantic Forms, Completeness and Coherence

In LFG subcategorization is defined in terms of subcategorizable and non-subcategorizable grammatical functions, semantic forms and global completeness and coherence well-formedness constraints on f-structures. Subcategorizable grammatical functions are those subcategorized for by some predicate, whereas non-subcategorizable grammatical functions are different types of adjuncts:

Subcategorizable GFs	Nonsubcategorizable GFs
`subj, obj, obj2, obl poss, comp, xcomp`	`adjuncts`

Semantic forms are the values of `pred` features and record the particular subcategorization requirements of predicates.[5] To give an example, the lexical entry for the active form of transitive `extinguish` would have the following semantic form:

$$\begin{array}{|l} V \quad \text{extinguish} \\ (\uparrow \text{PRED}) = \text{EXTINGUISH}\langle \uparrow \text{SUBJ}, \uparrow \text{OBJ} \rangle \end{array}$$

The global completeness and coherence constraints ensure that *all* the grammatical functions specified by some predicate are present in an f-structure governed by that predicate and that the *only* subcategorizable grammatical functions specified in that semantic form are present in the f-structure. The definitions are : an f-structure is *locally complete* iff it features all grammatical functions specified in the semantic form of its local PRED feature. An f-structure is *complete* iff it is locally complete and all its subsidiary f-structures are locally complete. An f-structure is *locally coherent* iff the only subcategorizable grammatical functions featured at that level are those listed in the semantic form value of the local PRED feature. An f-structure is *coherent* iff it is locally coherent and all its subsidiary f-structures are locally coherent.

3.2 Automatic Compilation of Semantic Forms

The f-structures constructed by grammar G1 mostly contain simple surface word forms as values of their `pred` features. In principle, proper semantic forms can be obtained in a number of ways: we can hand code them in the lexicon or we could try to derive them semi-automatically from information stored in a machine readable dictionary ([2]).

Given the f-structure resources generated by our treebank to f-structure compilation method there

[5] Semantic forms in LFG do more work than record subcategorization requirements. They are instrumental in linking syntactic to semantic arguments.

is a simple automatic way to obtain semantic forms. Consider the following tree bank entry

(13)
```
tree(a001,'43',v,sent(n(nn2(police)),v(vvd(checked),
n(at(the),nn1(coliseum)),p(if(for),n(nn2(bombs))),
p(ics(before),n(at(the),nn1(march))))))).
```

and the f-structure generated from it by our method:

(14)
```
subj : pred : police
       num  : pl
obj  : spec : the
       pred : coliseum
       num  : sg
obl  : obj  : pred : bombs
              num  : pl
       pred : for
vp_adjunct  : obj  : spec : the
                     pred : march
                     num  : sg
              pred : before
tense : past
pred  : checked
```

The semantic form associated with the lexical entry for `checked` at stake would be:

V		checked
$(\uparrow \text{PRED})$	$=$	$\text{CHECK}\langle\uparrow \text{SUBJ}, \uparrow \text{OBJ}, \uparrow \text{OBL}_{\text{FOR}}\rangle$
$(\uparrow \text{TENSE})$	$=$	PAST

Given the f-structure in (14) above, we can simply read off the required semantic form as follows: given the predicate `pred:checked` we simply collect all subcategorizable grammatical functions present at the same level of f-structure representation. If we apply this idea recursively throughout the f-structure we obtain semantic forms for all `pred:word` pairs present in the representation. This can be made explicit in terms of the following algorithm:

(15)
```
given a set of subcategorizable grammatical functions
   for each f-structure:
      for each level of embedding (in those f-structures):
         determine pred value and
         collect subcategorizable grammatical functions
```

Applied to the f-structure in (14) we obtain:

(16)
```
police([])  coliseum([])   bombs([])  march([])  for([obj])
before([obj]) checked([obj,subj,obl:for])
```

Note the following interesting parallel: ([3]) read off phrase structure rules from tree bank entries. In the semantic form compilation we employ the same methodology but simply "one level higher" in the hierarchy of linguistic representation.

In addition to compile semantic forms for particle verbs, we also compile particle information. Some examples from obtained from our corpus are:

(17)
```
turned([obj,subj,obl:pred:into],[]).
turned([obj,subj],[part:[pred:up|_]]).
turned([subj,obl:pred:for],[part:[pred:out|_]]).
```

Our implementation of completeness and coherence requires that any `part:pred` value specified in the `part` structure be present in the f-structure under consideration.

3.3 Folding Semantic Forms into the Grammar

Semantic forms and an implementation of completeness and coherence turn the grammar developed in experiment 1 into a stand-alone LFG resource. In order to automatically integrate the semantic forms compiled into our grammar we proceed as follows:

1. We automatically index the surface word form `pred` values by lexical tag, in order to account for categorial ambiguity. To give an example from our corpus, `records` can be a nominal or a verbal form with different semantic forms and during parsing we need to ensure the retrieval of the appropriate semantic form.
2. From this we generate indexed f-structures generated following the method described in experiment 1.
3. From this we compute indexed semantic forms.
4. In lexical access during parsing, given an input `word:tag` pair the relevant lexical macro automatically retrieves the appropriately indexed semantic form and maps that into the f-structure as the value of the `pred` feature under construction.

We then "reparsed" our treebank entries using proper semantic forms and the completeness and coherence check. Not surprisingly, for each of the 100 sentences we obtain a f-structure which is complete and coherent.[6] Note that even in the case of the deterministic grammar G1, inclusion of proper semantic forms introduces some limited non-determinism in "reparsing": a single word (even with the same lexical tag) can be associated with a number of semantic forms and these are systematically tried during "reparsing". The result of "reparsing" is, however, still deterministic, in that completeness and coherence filter out the non-determinism that happens during the "reparse".

4 Evaluation of Experiments 1 and 2

The semantic forms compiled are as good as the input f-structures. Put differently, the semantic forms compiled are an indicator of the quality of the f-structures obtained in experiment 1. The re-

[6] This contrasts with what we reported earlier in ([17]) and ([18]). The grammar developed at that stage in some cases allowed subcategorizable grammatical functions to be "stranded". This happened in cases where specifier `spec` functions act as head of `NP`s but did not introduce a `pred` function.

sults are surprisingly good. Below are some example semantic forms (verbal and nominal) compiled from the f-structures generated in experiment 1:

(18)
```
alert([obj,subj,obl:to],[])
bounced([subj,obl:off],[])
checked([obj,subj,obl:for],[])
comment([subj,obl:on],[])
estimated([obj,subj,obl:at],[])
fell([subj,obl:to],[])
joined([obj,subj,obl:in],[])
limited([subj,obl:by],[])
refrain([subj,obl:from],[])
talk([subj,obl:about],[])
```

```
coalition([obl:of],[])
declaration([obl:of],[])
investigation([obl:of],[])
members([obl:of],[])
pictures([obl:of],[])
possibility([obl:of],[])
reference([obl:to],[])
rules([obl:on],[])
tribute([obl:to],[])
use([obl:of],[])
```

The next collection shows semantic forms that point to aspects of our current grammar that can be improved upon:

(19)
```
 ...
claiming([obj,subj,obl:pred:in],[])
 ...
believe([comp],[])
believes([comp,subj],[])
 ...
```

Inspection of the semantic forms shows the following shortcomings in our current grammar G1:

1. `pred` values are mostly (tagged) surface word forms. This can result in "duplicate" semantic forms with identical subcat lists but different inflected forms of the predicate. This problem can be fixed by automatic morphological analysis to root form.

2. Our current grammar does not always distribute material into coordinate structures and relative clauses (subject and complement). This is reflected in semantic forms missing mostly subject functions.

3. Control and raising constructions are not identified lexically by the AP tag set. Again this is reflected in semantic forms missing mostly subject functions. This problem can be fixed by manually adding the required control equations to lexical entries, or, alternatively by additional rule annotations.

   ```
   lex(vvd(tried),FStr) :-
       FStr:pred === try,
       FStr:subj === FStr:xcomp:subj.
   ```

4. In some cases G1 gets the adjunct / subcategorized grammatical function distinction wrong.

Problem (2) can only be fixed properly by more sophisticated rule annotations exploiting the whole LFG constraint language mechanism such as $=_c$ constraint equations and functional uncertainty

constraints. In the absence of such, manual post-editing of semantic forms obtained is required. For English, most problems incurred in (2) and (3) can be fixed by automatically adding a subject `subj` function to semantic forms missing this function. Notice, however, that semantic forms need only be "repaired" in this fashion if they are to be used by an "ideal" LFG grammar. Of course, if G1 is used, as it stands, for parsing free strings, semantic forms as delivered by experiment 2 need to be employed and we need to accept slightly incomplete f-structures. Problem (4) can be approached by introducing some limited non-determinism into G1 and this is what we turn to next.

5 Limited Non-Determinism: Experiment 3

The deterministic approach reported in experiments 1 and 2 does not always get the distinction between adjuncts and subcategorized grammatical functions right. In some cases, what should be an adjunct is analyzed as a subcategorizable grammatical function or vice versa, simply because in the deterministic setup, given a particular rule a choice has to be made. In most cases this choice is correct since rules tend to be very specific and apply to few cases only. But even if that choice is correct for most cases it gives rise to unwanted semantic forms when it goes wrong. As an example, consider the following simple rule which expands a nominal `n(A)` into a sequence of a singular determiner `at1(B)`, followed by a singular common noun `nn1(C)` followed by some prepositional element `p(E)`. Assume that in most cases in the corpus where the rule applies `p` contributes an adjunct `np_adjunct`. In the deterministic approach this would lead to a single `A:np_adjunct=== E` equation in the rule annotation. It may turn out, however, that in one or two cases `p(E)` contributes an oblique object. Of course, this can be captured by introducing a disjunction `( A:obl === E ; A:np_adjunct === E )` as follows:

(20)
```
rule(n(A), [at1(B),nn1(C),p(E)]) :-
   A === B,
   A === C,
   ( A:obl === E ; A:np_adjunct === E ).
```

It turns out that there are only 8 offending rules in G1 that need to be made non-deterministic in the way described in (20) to yield a new grammar G2.

Given G2 we now compile a new set of f-structures by "reparsing" our treebank fragment. The amount of non-determinism generated turns out to be very limited:

#sentences.	#analyses
67	1
27	2
6	4

In the cases where there are multiple f-structures for an input treebank item, we manually select the best analysis for inclusion in the corpus. From this corpus we then recompute semantic forms as detailed in the description of experiment 2.

Again we automatically fold the semantic forms obtained in experiment 3 into our non-deterministic grammar G2 and "reparse" our original treebank with completeness and coherence constraints enforced. The results are shown below:

#sentences	#analyses
93	1
7	2

All 100 sentences are associated with f-structures satisfying completeness and coherence (see again footnote 7.) 7 sentences are associated with 2 f-structures each of which is complete and coherent.

6 Structure Preserving Grammar Compaction: Experiment 4

LFG c-structure components developed manually by linguists tend to be compact, i.e. they have a small number of rules, each of which is designed to express maximal generalizations. Compared to this, the CF-PSG compiled automatically from our treebank resource is of "poor quality": it contains a large number of highly specific rules, most of which apply to very few cases. To a certain extent this is due to the fact that the treebank rules are compiled from a corpus of real language, as opposed to text book grammars designed to illustrate a number of well chosen points. This said, the AP tag set contains a high number of lexical and non-lexical tags (grammatical categories) with fine-grained distinctions. This entails a large number of often only minimally different rules with often substantial left common prefixes. There is very little use of recursion and no general X-bar theoretic structuring approach is discernible in the rules. In fact, the CF-PSG obtained from the tree bank looks much like a unification grammar compiled out into monadic categories. While to a certain extent this situation is conducive to our deterministic grammar annotation experiment in that the the fine grained information is exploited in our macros annotations, it also results in substantial repetition overhead in the annotation effort. Ideally, we would like to obtain a CF-PSG from our treebank entries which is somewhat more in line with current LFG practice.

In a recent paper ([11]) develop a very interesting automatic grammar compaction methodology. The basic idea is the following: rules whose RHS's can be parsed by other rules (yielding the same LHS) are replaced by those rules. To give a simple example, consider the following grammar fragment:

(21)
```
VP -> V NP PP        (1)
VP -> V NP           (2)
NP -> NP PP          (3)
```

The first rule can be replaced by rules `(2)` and `(3)` and the resulting grammar will still accept the same strings. However, the grammar transformation is not structure preserving. In fact, in the new grammar, every flat tree of the form `VP(V,NP,PP)` will be replaced by a tree of the form `VP(V,NP(NP,PP))`:

(22)
```
VP(V,NP,PP) => VP(V,NP(NP,PP))
```

Hence this grammar compaction technique is not guaranteed to respect linguistically motivated structure. Fortunately, there is another possibility open to us. The basic idea is very simple: earlier we remarked that the CF-PSG obtained from the tree bank looks much like a unification grammar compiled out into monadic categories. Our annotations in the form of lexical macros and rule

annotations exploit the multitude of categories and, in a sense, shift information from the complex categories "back" into f-structures. It is in this sense that we "repackage" information. This means that once we have shifted the information gleaned from tags into feature structures, the original tags have done their work and we can collapse the original tags into a smaller set of "supertags". Consider the following fragment picked from the grammar compiled from the original treebank entries:

(23)
```
rule(n(_), [at(_),jb(_),nn1(_)]).                    (1a)
rule(n(_), [at(_),jj(_),nn1(_)]).                    (2a)
...
rule(n(_), [at1(_),jb(_),nn1(_)]).                   (1b)
rule(n(_), [at1(_),jj(_),nn1(_)]).                   (2b)
```

Rules `(1a)` and `(2a)` differ only with respect to the second element (different type of adjective `jj` and `jb`) on their RHSs. So do `(1b)` and `(2b)`. Furthermore, all `(a)` rules differ from the `(b)` rules only in the first element on their RHS. In the `(a)` rules we have an article `at(_)` not specified for number, while the `(b)` rules require singular articles `at1(_)`. Suppose now that lexical macros and rule annotations have shifted this information into f-structure, then we can collapse the original adjective tags `jj` and `jb` into an adjective supertag `adj` and the article tags `at(_)` and `at1(_)` into a determiner supertag `det(_)`. This means we collapse the *four* rules `(1a)`, `(2a)`, `(1b)` and `(2b)` into a *single* general rule:

(24)
```
rule(n(_), [det(_),adj(_),nn1(_)]).
```

Notice that in contrast to ([11]) our grammar compaction method is structure preserving. We merely relabel nodes and never change number or structural relations between nodes in trees. This means that the relabeled and the original trees are isomorphic up to node labeling. We automate this type of grammar compaction as follows:

1. We automatically compile out lexical macros over the lexical entries, thereby associating f-structures with words.

2. We use our linguistic knowledge to collapse the tagset using "supertags", e.g.:

```
DD1     singular determiner (this, that, another)
DD2     plural determiner (these, those)
DDQ     wh-determiner (which, what)
DDQZ    wh-determiner, genitive (whose)
DDQV    wh-ever determiner (whichever, whatever)

collapse(dd1,det).
collapse(dd2,det).
collapse(ddq,det).
collapse(ddqz,det).
collapse(ddqv,det).
```

3. Then we automatically relabel the lexicon using the `collapse(Old,New)` information.

4. We automatically relabel the treebank in exactly the same fashion.

5. We recompile a collapsed CF-PSG from the tree-using the method of ([3]).

6. We annotate the collapsed CF-PSG by "merging" (inheriting) functional annotations from the original CF-PSG to the collapsed rules. Since our method is structure preserving it will not change the length of the RHS of grammar rules. This allows us to identify rules which have been collapsed. Notice that merging annotations from different rules introduces disjunctions.

7. We can then "reparse" the tree bank entries using the new collapsed and annotated grammar using completeness and coherence.

Results grammar compaction:

(25)

original grammar	511 rules
compacting lexical tags	367 rules
compacting lexical and phrasal tags	332 rules

7 Discussion and Further Work

In the present paper we have outlined a method for semi-automatic LFG corpus construction. As input the method requires a treebank. From the treebank a CF-PSG is induced automatically. Manual annotation of the derived grammar rules together with lexical macros define an initial LFG grammar. Finally, the original treebank representations (the annotated trees - not the strings) are "reparsed" using the induced LFG grammar. "Reparsing" simply follows the c-structure annotations of the original treebank and induces the corresponding f-structure. The method avoids much of the non-determinism (in c-structure parsing) incurred by alternative approaches. We have shown how our method can be extended to compile a stand-alone LFG resource. This involved the development of semi-automatic compilation of semantic forms and a simple structure preserving grammar compaction technique.

We believe that the methodology presented is of a general nature. In addition to LFG, we hope to port it successfully to other decorated grammar formalisms such as GPSG, CUG and perhaps even HPSG inspired approaches and as well as to different treebanks.

The methodology described in the present paper involves a manual annotation part. At the moment we are experimenting with automatic annotation schemes. We compile annotation templates and apply these automatically to CF-PSG fragments. We hope to be able to report on the results of these experiments in the near future. It is important to scale up our experiments by at least an order of magnitude, particularly so if at some stage the resources produced are to serve as training corpora for statistical approaches to LFG. Automatic annotation can make a significant contribution to handling larger rule sets.

There are many interesting possibilities to extend the research described in the present paper and we hope to be able to follow and report on some of them in the future:

- To date, our work has focussed very much on interesting engineering issues. The grammar annotation task, however, has revealed many interesting linguistic issues in applying LFG to real text. We would like to be able to address some of these.

- Some treebanks (e.g. Susanne) come with basic functional annotations. It would be interesting to explore automatic compilation of partially annotated LFG grammars from these.
- Mixed scenarios are conceivable: take an existing large scale LFG and a treebank. Automatically relabel the treebank (using the method described in our grammar compaction experiment) and manually adjust some of the rules in the LFG to fit treebank and LFG grammar. Then apply our "reparsing" idea (which is deterministic on the c-structure) to map the treebank into a feature structure annotated treebank.
- We would like to test the stand-alone LFG grammars in free parsing.
- Our simple structure preserving grammar compaction holds considerable potential for further research: it is a grammar induction technique which goes from many specific rules to fewer, more general ones. It could be approached fruitfully from a machine learning perspective. Furthermore, and in a slightly different direction, one could try to combine our grammar compaction technique with the methodology developed by ([11]). It is well known that treebank grammars keep growing with the size of the corpus ([4]). It would be interesting to see what effect a combined approach would have on the size and growth factor of rule sets. In such a scenario, our technique should simply be applied with brute force to map a monadic CF-PSG into another monadic CF-PSG, i.e. independently of whether or not information encoded in the fine grained category distinctions has been lost. Other compatible approaches would include finding regular (!) regularities in RHS of (collapsed and compacted) grammar rule sets.

Acknowledgements

Carl Vogel commented extensively on an early draft. Many thanks also to the participants, particularly to Thorsten Brants and John Carroll, of the LINC-99 and Atala Treebank workshops in Bergen and Paris, June and July 1999, where earlier versions and fragments of our work ([17, 18]) were presented. Mistakes and opinions are our own.

References

[1] R. Bod and R. Kaplan. A Probabilistic Corpus-Driven Model for Lexical-Functional Analysis. In COLING: *Proceedings of the 17th International Conference on Computational Linguistics & 36th Conference of the Association for Computational Linguistics*, Montreal, Canada, **1**:145–151, 1998.

[2] K. Brazil. Building Subcategorisation Lexica for an LFG Grammar of French. Technical Report. Grenoble: Xerox Research Centre Europe. 1997

[3] E. Charniak. Tree-bank grammars. In AAAI-96, Proceedings of the Thirteenth National Conference on Artificial Intelligence, MIT Press, pp.1031-1036.

[4] E. Charniak. Statistical Techniques for Natural Language Parsing. *AI Magazine* **18**:4, pp.33–44, AAAI, Menlo Park, California, 1997.

[5] A. Frank, T.H. King, J. Kuhn and J. Maxwell. Optimality Theory Style Constraint Ranking in Large-scale LFG Grammars. In LFG'98, Proceedings of the Conference, The University of Queensland, Brisbane, Australia, pp.174-189. CSLI Publications, http://www-csli.stanford.edu/publications/.

[6] R. Garside, G. Leech and T. Varadi (compilers). The Lancaster Parsed Corpus. A machine-readable syntactically-analysed corpus of 144,000 words, available for distribution through ICAME, The Norwegian Computing Centre for the Humanities, Bergen, 1992.

[7] G. Gazdar and C. Mellish. Natural Language Processing in Prolog: an Introduction to Computational Linguistics. Addison-Wesley, Wokingham, UK, 1989.

[8] H. van Halteren and N. Oostdijk. Towards a Syntactic Database: the TOSCA Analysis System. In: J. Aarts, P. de Haan & N. Oostdijk (eds), *English Language Corpora: Design, Analysis and Exploitation*, Rodopi, Amsterdam, pp.145–162, 1993.

[9] Charles T. Hemphill, John J. Godfrey and George R. Doddington. The ATIS Spoken Language Systems Pilot Corpus. In *Proceedings of the DARPA Speech and Natural Language Workshop*, Hidden Valley, Pa., pp.96–101, 1990.

[10] R.M. Kaplan and J. Bresnan. Lexical functional grammar. In Bresnan, J., (editor), *The mental representation of grammatical relations.* MIT Press, Cambridge Mass. 173–281, 1982.

[11] A. Krotov, M. Hepple, R. Gaizauskas and Y. Wilks. Compacting the Penn Treebank Grammar. In COLING-ACL'98, Proceedings of the Conference, Volume I, Montreal, Quebec, Canada, pp.699-703.

[12] G. Leech & E. Eyes. Syntactic Annotation: Treebanks. In: R. Garside, G. Leech & A. McEnery (eds), *Corpus Annotation: Linguistic Information from Computer Text Corpora*, Addison-Wesley/Longman, Harlow, UK, 1997.

[13] G. Leech and R. Garside. Running a Grammar Factory: on the Compilation of Parsed Corpora, or 'Treebanks'. In: S. Johansson and A.-B. Stenström (eds), *English Computer Corpora: Selected Papers and Research Guide*, Mouton de Gruyter, Berlin, pp.15–32, 1991.

[14] M.T. López-Soto and M.G. Fernández-Díaz. Integration of Semantic Patterns and Statistical Information for an LFG-based Parser. In LFG'96, Proceedings of the Conference, Rank Xerox Research Centre, Grenoble, France, pp249-261. CSLI Publications, http://www-csli.stanford.edu/publications/.

[15] M. Marcus, B. Santorini and M. Marcinkiewicz. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics* **19**:2, pp.313–330, 1993.

[16] G. Sampson. English for the Computer: the SUSANNE Corpus and Analytic Scheme. Clarendon, Oxford, 1995.

[17] J. van Genabith, L. Sadler and A. Way Data-Driven Compilation of LFG Semantic Forms EACL 99, Workshop on Linguistically Interpreted Corpora (LINC-99), Bergen, Norway, June 12th, 1999, pp.69-76.

[18] J. van Genabith, L. Sadler and A. Way Structure Preserving CF-PSG Compaction, LFG and Treebanks Journées ATALA, Corpus annotés pour la syntaxe, Actes, (Proceedings ATALA Workshop - Treebanks), l'Université Paris 7, France, 18-19 Juin 1999, pp.107-114, also `http://talana.linguist.jussieu.fr/treebanks99/`