Treebank vs. Xbar-based Automatic F-Structure Annotation

Josef van Genabith
Computer Applications
Dublin City University
Dublin, Ireland
josef@compapp.dcu.ie

Anette Frank
Language Technology Group
DFKI GmbH
Saarbrücken, Germany
frank@dfki.de

Andy Way
Computer Applications
Dublin City University
Dublin, Ireland
away@compapp.dcu.ie

Proceedings of the LFG01 Conference

University of Hong Kong, Hong Kong

Miriam Butt and Tracy Holloway King (Editors)

2001

CSLI Publications

http://csli-publications.stanford.edu

Treebank vs. Xbar-based Automatic F-Structure Annotation

Abstract

Manual, large scale (computational) grammar development is time consuming, expensive and requires lots of linguistic expertise. More recently, a number of alternatives based on treebank resources (such as Penn-II, Susanne, AP treebank) have been explored. The idea is to automatically "induce" or rather read off (P)CFG grammars from the parse annotated treebank resources and to use the treebank grammars thus obtained in (probabilistic) parsing or as a starting point for further grammar development. The approach is cheap, fast, automatic, large scale, "data driven" and based on real language resources. Treebank grammars typically involve large sets of lexical tags and non-lexical categories as syntactic information tends to be encoded in monadic category symbols. They feature flat rules (trees) that can "underspecify" attachment possibilities. Treebank grammars do not in general follow Xbar architectural design principles (this is not to say that treebank grammars do not have design principles). As a consequence, treebank grammars tend to have very large CFG rule bases (e.g. Penn-II $>$ 17,000 CFG rules for about 1 million words of text) with often only minimally differing rules. Even though treebank grammars are large, they are still incomplete, exhibiting unabated rule accession rates. From a grammar engineering point of view, the size of the rule base poses problems for maintainability, extendability and, if a treebank grammar is to be used as a CF-base in a LFG grammar, for functional (feature-structure) annotations. From the point of view of theoretical linguistics, flat treebank trees and treebank grammars extracted from such trees do not express linguistic generalisations. From the perspective of empirical and corpus linguistics, flat trees are well-motivated as they allow underspecification of subtle and often time consuming attachment decisions. Indeed, it is sometimes doubted whether highly general Xbar schemata usefully scale to "real" language. In previous work we developed methodologies for automatic feature-structure annotation of grammars extracted from treebanks. Automatic annotation of "raw" treebank grammars is difficult as annotation rules often need to identify subsequences in the RHSs of flat treebank rules as they explicitly encode head, complement and modifier relations. Xbar-based CFG rules should substantially facilitate automatic feature-structure annotation of grammar rules. In the present paper we conduct a number of experiments to explore a space of possible grammars based on a small fragment of the AP treebank resource. Starting with the original treebank fragment we automatically extract a CFG G. We then apply an automatic structure preserving grammar compaction step which generalises categories in the original treebank fragment and reduces the number of rules extracted, resulting in a generalised treebank fragment and in a compacted grammar G^c. The generalised fragment is then manually corrected to catch missed constituents (and the like) resulting in an automatically extracted, compacted and (effectively manually) corrected grammar $G^{c,m}$. Manual correction proceeds in the "spirit" of treebank grammars (we do not introduce Xbar analyses). We then explore how many of the manual correction steps on treebank trees can be achieved automatically. We develop, implement and test an automatic treebank "grooming" methodology which is applied to the generalised treebank fragment to yield a compacted and automatically corrected grammar $G^{c,a}$. Grammars $G^{c,m}$ and $G^{c,a}$ are very similar to compiled out "flat" LFG-82 style grammars. We explore regular expression based compaction (both manual and automatic) to relate $G^{c,m}$ to a LFG-82 style grammar design. Finally, we manually recode a subsection of the generalised and manually corrected treebank fragment into "vanilla-flavour" Xbar based trees. From these we extract a compacted, manually corrected, Xbar-based grammar $G^{c,m,x}$. We evaluate our grammars and methods using standard labelled bracketing measures and according to how well they perform under automatic feature-structure annotation tasks.

1 Introduction

Manual, large scale (computational) grammar development is time consuming, expensive and requires lots of linguistic expertise. More recently, a number of alternatives based on treebank resources (such as Penn-II, Susanne, AP treebank) have been explored. The idea is to automatically "induce" or rather read off (P)CFG grammars from the parse annotated treebank resources and to use the treebank grammars thus obtained in (probabilistic) parsing [Charniak,93] or as a starting point for further grammar development [Krotov et al,98]. The approach is cheap, fast, automatic, large scale, "data driven" and based on real language resources.

Treebank grammars typically involve large sets of lexical tags and non-lexical categories as syntactic information tends to be encoded in monadic category symbols. They feature flat rules (trees) that can "underspecify" attachment possibilities. Treebank grammars do not in general follow Xbar architectural design principles (this is not to say that treebank grammars do not have design principles). As a consequence, treebank grammars tend to have very large CFG rule bases (e.g. Penn-II $>$ 17,000 CFG rules for about 1 million words of text) with often only minimally differing rules. Even though treebank grammars are large, they are still incomplete, exhibiting unabated rule accession rates. From a grammar engineering point of view, the size of the rule base poses problems for maintainability, extendability and, if a treebank grammar is to be used as a CF-base in a LFG grammar, for functional (feature-structure) annotations. From the point of view of theoretical linguistics, flat treebank trees and treebank grammars extracted from such trees do not express linguistic generalisations. From the perspective of empirical and corpus linguistics, flat trees are well-motivated as they allow underspecification of subtle and often time consuming attachment decisions. Indeed, it is sometimes doubted whether highly general Xbar schemata usefully scale to "real" language.

In previous work [Sadler, van Genabith and Way,00] and [Frank,00] developed methodologies for automatic feature-structure annotation of grammars extracted from treebanks (see also [Frank et al,01]). Automatic annotation of "raw" treebank grammars is difficult as annotation rules often need to identify subsequences in the RHSs of flat treebank rules. Xbar-based CFG rules should substantially facilitate automatic feature-structure annotation of grammar rules as they explicitly encode head, complement and modifier relations [Bresnan,01].

In the present paper we report on a number of experiments in which we explore a space of possible grammars based on a small fragment of the AP treebank resource.

Starting with the original treebank fragment we automatically extract a CFG G. We then apply an automatic structure preserving grammar compaction step which generalises categories in the original treebank fragment and reduces the number of rules extracted, resulting in a generalised treebank fragment and in a compacted grammar G^c. The generalised fragment is then manually corrected to catch missed constituents (and the like) resulting in an automatically extracted, compacted and (effectively manually) corrected grammar $G^{c,m}$. Manual correction proceeds in the "spirit" of treebank grammars (we do not introduce Xbar analyses). We then explore how many of the manual correction steps on treebank trees can be achieved automatically. We develop, implement and test an automatic treebank "grooming" methodology which is applied to the generalised treebank fragment to yield a compacted and automatically corrected grammar $G^{c,a}$. Grammars $G^{c,m}$ and $G^{c,a}$ are very similar to compiled out "flat" LFG-82 style grammars. We explore regular expression based compaction (both manual and automatic) to relate $G^{c,m}$ to a LFG-82 style grammar design. Finally, we manually recode a subsection of the generalised and manually corrected treebank fragment into "vanilla-flavour" Xbar-based trees. From these we extract a compacted, manually corrected, Xbar-based grammar $G^{c,m,x}$.

We evaluate our grammars and methods using standard labelled bracketing measures and according to how well they perform under automatic feature structure annotation tasks.

2 Automatic F-Structure Annotation

F(eature)-structures [Kaplan and Bresnan,82] are "higher level", abstract syntactic representations. It would be desirable to have treebank resources with both CF-PSG and feature-structure representations

- as training resources for probabilistic LFG/unification/constraint-based grammars
- to develop stand-alone LFG/unification/constraint-based grammar resources

In previous work [Sadler, van Genabith and Way,00] and [Frank,00] developed two methodologies for automatically annotating treebank trees and grammars extracted from such trees with feature-structure information. One method involves the statement of regular expression based feature-structure annotation principles, the other (cascaded) rewriting of tree descriptions in a tree description language.

2.1 Regular Expression Based Annotation

Regular expression based annotation is best explained by way of example. The following is a flat CFG rule extracted from a treebank resource (note that each constituent is of the form `cat:F` where `cat` is a monadic CFG category and `F` is a variable designed to carry feature-structure information):

```
vp:VP > adv:A v0:V0 v0:V1 v0:V2 s:S pp:P
```

In flat treebank rules such as this, RHSs often correspond to sequences of multiple (embedded) constituents. As a consequence, annotation principles need to be partial and underspecified in order to be able to associate subsequences of RHSs with the required feature-structure annotations. We employ regular expression to express underspecified rule LHSs and RHSs and to pick out subsequences in rule RHSs. Annotation principles are of the form `LHS > RHS @ ANN`, where `ANN` is a set (in our implementations a list) of conjunctions or disjunctions of feature-structure equations, such as:

```
vp:VP > * v0:V1 v0:V2 * @ [V1:xcomp=V2,V1:subj=V2:subj].

vp:VP > *(~v0) v0:V0 *  @ [VP=V0].

vp:VP > * v0:V0 s:S *   @ [V0:comp=S].

        :                 :
```

The first principle identifies `v0:V1 v0:V2` subsequences in rule RHSs and states that for two adjacent `v0`s the feature structure associated with the second `v0` provides the value of an `xcomp` feature of the first. Additionally, both share the same value for their `subj` attributes. The second principle states that the leftmost `v0` is the head of a `vp` rule. A particular principle may apply several times to different subsequences in the RHS of a single rule and a number of different principles may apply to any one rule. All principle applications are collected and the rule is annotated accordingly:

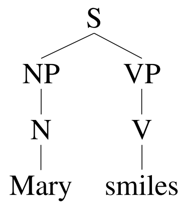
```
vp:VP > adv:A v0:V0 v0:V1 v0:V2 s:S pp:P
      @ [VP=V0, V0:xcomp=V1, V0:subj=V1:subj, V1:xcomp=V2,
         V1:subj=V2:subj, V2:comp=S].
```

2.2 Tree Annotation Using Tree Descriptions

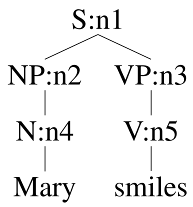
In our second method annotation principles are applied to partial configurations of treebank *trees* using a tree rewriting technique. Again, we illustrate the approach by way of a simple example.

Treebank trees are first encoded as a flat collection (conjunction) of basic tree description predicates (`arc, prec, lex`) in a tree description language. Tree nodes are associated with a unique identifier (`n1,n2,..`).

```
[s,[np,[n,Mary]],
   [vp,[v,smiles]]].
```




```
⇒   arc(n1,s,n2,np), arc(n2,np,n4,n)
    arc(n1,s,n3,vp), arc(n3,vp,n5,v),
    prec(n2,n3),
    lex(n4,Mary), lex(n5,smiles)
```

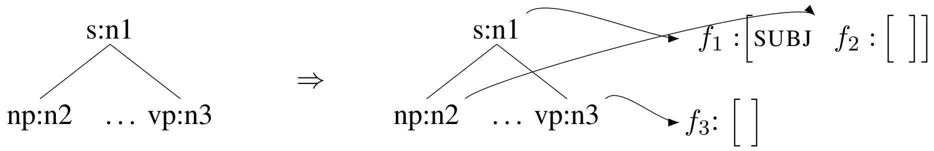


We then state annotation principles in terms of rewriting rules which apply to the flat set of tree descriptions. Rewriting rules are of the form `CD` $\Longrightarrow$ `AD` where `CD` is a constraining tree description and `AD` is an annotation description. The formalism supports a set of switches that optionally allow consumption of constraining tree descriptions during rule application. Annotation principles can thus support both order independent (enriching) and cascaded order dependent (rewriting) annotation.

In the case at hand the constraining tree configuration is provided by a local `s > np vp` subtree. The annotation description introduces new tree description predicates which associate each tree node with a feature-structure via the `phi` projection. The feature-structure associated with the subject `np` is constrained to be the value of the `subj` attribute of the feature-structure associated with the sentence node.

Unlike the regular expression based annotation method which is restricted to (partial) local trees of depth one, the description-based approach can access (partial) non-local tree configurations.

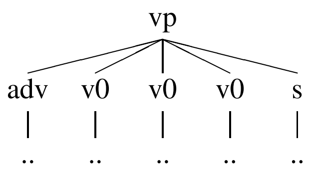
```
+arc(A,s,B,np),
+arc(A,s,C,vp), +prec(B,C)   ==>   phi(A,FA), phi(B,FB), subj(FA,FB).
```



In the present paper we will use the regular expression based method operating on PS rules for specifying feature-structure annotation principles and the tree description based rewriting approach for our automatic treebank "grooming" methodology presented in Section 7 below.

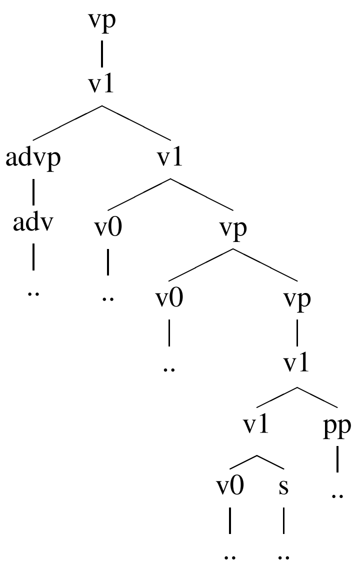
3 Flat vs. Xbar-based Treebank Rules and Annotation

Consider again the flat treebank example rule here displayed as a local tree configuration:



Automatic annotation can be difficult as annotation principles have to identify suitable subsequences on the RHS of the corresponding CFG rule for annotation. This may not always be possible resulting in partial and/or incorrect annotations.

Contrast this with a "vanilla flavour" Xbar recoding of the tree (we will say more in Section 9 below about what exactly we mean by "vanilla flavour" Xbar encoding):



The recoding is exclusively binary and unary branching with each local subtree clearly identifying head, adjunct and complement relations. The Xbar configuration and grammar design principles should guide and facilitate automatic feature-structure annotation to a much greater extent than the flt treebank rule representations which often conflate otherwise embedded subconstituents. The rule base for the recoded tree consists of:

```
vp > v1       v1 > advp v1    v1 > v1 pp
vp > v0 vp    v1 > v0 vp      v1 > v0 s
```

The corresponding annotation principles are:

```
__:X > __:Y       @ [X=Y]
v1:X > v1:Y  _:Z @ [X=Y,X:adjn $ Z]
v1:X >  _:Z v1:Y @ [X=Y,X:adjn $ Z]
v1:X > v0:Y  s:Z @ [X=Y,X:comp Z]
    ...
```

The first principle states that any unary branching rules project their feature-structures. The second and third identify the head in pre- and post-modifier configurations and make the modifiers elements '$' of an adjunct set. Finally, a sentential complement provides the value of the complement attribute `comp`. Notice that in the case at hand, f-structure annotation is completely driven by Xbar configuration. Notice further that in contrast to our original annotation principles, for the flat treebank configuration, Xbar-based annotation principles are no longer underspecified. There is no need to identify subsequences of constituents as a separate constituent in RHSs of binary branching Xbar rules.

Intuitively, Xbar-based automatic annotation looks attractive. At this stage we may summarise our preliminary findings in the following table:

Xbar	TreeBank
principle based architecture	underspecification and pragmatism
generalisations	many flat rules
small rule base	large rule base
conducive to automatic annotation	difficult for automatic annotation

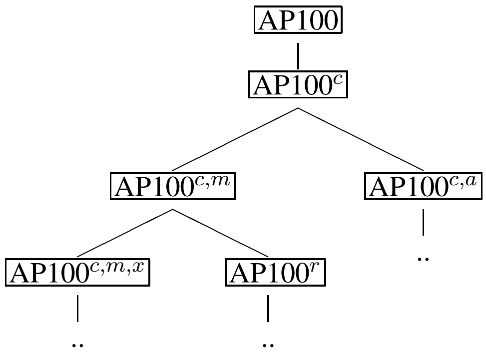
However, there are a number of considerations (often expressed by practitioners working in the fields of empirical and corpus linguistics) that sound a more cautionary note:

- Xbar design principles often force subtle and difficult attachment decisions and sometimes it may not even be clear how these can be resolved. Cases in point are modifier attachment. Binary branching rules force attachment decisions which are not always clear cut in the case of multiple modifiers, pre- and post-modifiers of the same bar one level node or multiple attachment possibilities for a single modifier (e.g. where does the `pp` in our recoded example rule attach?). In any case, subtle and difficult decisions such as these are too time consuming and thus not practical for treebanking purposes. By contrast, flat treebank representations support (albeit in a somewhat crude fashion) a notion of underspecification not provided by classical Xbar design principles.

- Somewhat more speculatively it is sometimes assumed that Xbar design principles do not really scale up to real language in any useful way.

4 Treebank Grooming

In the present paper we report on a number of experiments to evaluate a "vanilla flavour" Xbar design approach to a fragment of the AP treebank to measure performance under automatic feature structure annotation and to ascertain the feasibility of Xbar inspired approaches to tree banking tasks and resources derived from treebanks. Starting with the original treebank fragment we move the fragment in stages to a resource inspired by Xbar design principles. Some of the moves are automatic and involve novel treebank "grooming" methodologies; others are manual. The different stages (and their interdependencies) are best displayed by way of a family tree:



Our experiments are based on the publicly available subsection AP100 (the first 100 parse-annotated sentences) of the AP treebank [Leech and Garside,91]. From this we automatically extract a grammar G following the method of [Charniak,93].[1] As our starting point we apply an automatic, structure preserving grammar compaction technique [van Genabith, Sadler and Way,99], [Hepple and van Genabith,00] to AP100 and generate AP100^c. The technique generalises overspecific categorisation and this can have the effect of collapsing grammar rules. From AP100^c we extract a compacted grammar G^c. We manually annotate G^c with feature-structure annotations to create a "gold-standard" against which we evaluate automatic feature-structure annotation. We then develop a set of regular expression based feature-structure annotation principles and apply them automatically to an unannotated version of G^c. We evaluate the results of automatic annotation against the gold standard using precision and recall measures [Sadler, van Genabith and Way,00]. These results provide the base line for our further experiments.

In the next step, we manually correct any "obvious" errors in AP100^c. These include missed groupings of constituents, attachment errors, relabelling of certain nodes and changes to the original representation of punctuation. All changes are in the spirit of the original treebank grammar (i.e. we do not introduce Xbar assumptions at this stage). Manual correction transforms AP100^c into AP100^{c,m}. From this we extract

[1]All the grammars in our experiments are automatically extracted from the original or "groomed" parse-annotated treebank resources in this fashion. The basic idea is simple: a recursive procedure traverses treebank trees and for each local tree of depth one records a corresponding CFG rule. CFG rule token counts allow the computation of simple relative frequency based maximum likelihood estimators to compute probabilistic versions of the CFGs extracted.

grammar $G^{c,m}$. We manually[2] annotate the corrected $G^{c,m}$ to produce a gold standard against which to evaluate automatic feature-structure annotation of $G^{c,m}$. We then develop a set of regular expression based feature-structure annotation principles and apply them automatically to an unannotated version of $G^{c,m}$. We evaluate the results of automatic annotation against the gold standard using precision and recall measures. The expected outcome is, of course, that a manually corrected grammar is "cleaner", more systematic and contains fewer "quirks" than the orginal treebank grammar and is thus easier to annotate automatically. We also report labelled bracketing measures to compare the structure of the trees in AP100^c to the manually "groomed" AP100^{c,m}.

In our next experiment, we investigate to which extent the manual transformation operations required to "groom" or "convert" the trees in AP100^c into AP100^{c,m} can in fact be performed automatically. Automatic treebank grooming is important as manual correction is extremely time consuming, error prone and risks missing analysis "bugs" in some of the source trees. We apply automatic treebank conversion techniques, which are based on the description based tree rewiting methodology presented in Section 2.2. above. A similar approach to treebank conversion is presented in [Frank,01]. We apply this technique to AP100^c to generate an automatically converted ("groomed") resource AP100^{c,a}. We compare the manually groomed AP100^{c,m} with the automatically converted AP100^{c,a} in terms of standard labelled bracketing measures and precision and recall results on transformation events.

Our next move is based on the observation that grammars $G^{c,m}$ and $G^{c,a}$ extracted from both the manually and the automatically groomed AP100^{c,m} and AP100^{c,a} look very much like compiled out (expanded) versions of flat LFG-82 [Kaplan and Bresnan,82] style grammars which involve regular expressions (such as optionality or Kleene-star) in CFG rule RHSs. We conduct two experiments to relate $G^{c,m}$ to a flat LFG-82 style grammar. The first is based on FST machinery: given a LHS we collect the corresponding RHSs in the grammar and union them into a regular expression. We then use FST machinery to compile the regular expression into a FSA and apply minimisation. The resulting net is then retranslated into a regular expression. In the second experiment, for a given LHS we hand code a regular expression based rule RHS based on the attested RHSs in the grammar. We verify that the language defined by the regular expression includes the language of the attested RHSs. Regular expression based rule RHSs can provide compact representations of CFG resources facilitating automatic (and manual) feature-structure annotation. For reasons of space, here we can only briefly outline our experiments. We hope to report on the regular expression based "compactions" elsewhere.

Finally, we manually recode the first 50 parse-annotated sentences from the generalised and manually corrected AP100^{c,m} using "vanilla flavour" Xbar design principles in AP50^{c,m,x}. From this we extract a grammar $G^{c,m,x}$. In all the grammars extracted in our experiments, the NP fragments pose the largest, most varied and complicated annotation tasks. We extract the NP fragment from AP50^{c,m,x}. We manually construct gold-standard feature-structure annotations for the Xbar-based NP fragment and develop regular expression based feature-structure annotation principles for the NP fragment. We evaluate the results of automatic annotation against the gold standard in terms of precision and recall measures. We also compare CFG rule growth rates for AP100^c and AP50^{c,m,x}.

5 Structure Preserving Grammar Compaction

Treebanks tend to express considerable amounts of information in monadic category labels.[3] Typically, this results in large lexical tag sets and in a large number of non-lexical categories. The AP treebank features

lexical tags	$>$	180
non-lexical cats	$>$	50

[2] Actually, to a large extent the gold standard annotation of $G^{c,m}$ is done automatically: the majority of the CFG rules in $G^{c,m}$ are identical to rules in G^c. The corresponding annotations from the already available gold standard G^c are moved automatically to the manually corrected $G^{c,m}$ and it is only the residue $G^{c,m} - G^c$ that needs to be annotated manually to create the gold standard feature-structure annotations for $G^{c,m}$.

[3] In addition, some treebanks allow adornment of CFG nodes in terms of a simple feature decoration mechanism, e.g. Penn-II supports features indicating grammatical functions or coindexation.

This, together with the flat treebank analyses, results in many often only minimally different CFG rule types. From the AP100 fragment we extract 509 rule types,[4] that is more than 5 distinct types per tree. (Automatic) feature-structure annotation repackages information: often - particularly so in the case of lexical entries - it "copies" information originally encoded in monadic category labels into feature-structure annotations (and hence the resulting feature-structure representations). As a consequence we can collapse highly discriminating tags into "supertags". In the original tag set, number information in nominals is encoded in terms of two tags `nn1` and `nn2` for singular and plural common nouns, respectively. Once number information is represented in terms of a feature value pair in a feature-structure description `num=sg` we can collapse the two orginal tags into a supertag `n0`

```
{ nn1,nn2 } => n0
```

Collapsing tags into supertags can have the effect of collapsing grammar rules. Consider the following scenario where `jj` and `jb` are two types of adjectives:

```
n > at1 jb nn1
n > at1 jj nn1
```

Collapsing `{ jj,jb } => adj` has the effect of collapsing the two CFG rules expanding `n` into the single rule

```
 n > at1 adj nn1
```

Structure preserving grammar compaction can be applied automatically to parse-annotated treebank resources and compacted grammars can then be extracted [van Genabith, Sadler and Way,99], [Hepple and van Genabith,00]. In previous work [Sadler, van Genabith and Way, 00] applied structure preserving compaction to AP100 generating AP100^c and extracted the corresponding grammars G (from AP100) and G^c (from AP100^c) with the following number of rule types:

	rule types
G	509
G^c	331

In order to evaluate automatic feature-structure annotation in terms of a gold standard, G^c was first annotated manually. This was followed by the development of regular expression based annotation principles. These annotation principles are then applied automatically to an unannotated version of G^c.

Automatic annotation is evaluated against the gold standard in terms of precision and recall results. Precision measures how many of the automatically generated annotations are correct (i.e. are in the gold standard); recall measures how many of the target gold standard annotations were in fact generated automatically:

	precision	recall
G^c	93.4 %	91.6 %

Automatic annotation is more partial than incorrect. The results for the compacted grammar G^c provide the base line for our further treebank grooming experiments reported below.

[4] As opposed to rule tokens.

6 Manual Correction

In the following sections we move the automatically generalised (with respect to category labels) treebank fragment to an Xbar-based encoding in stages. The first step is a manual correction/clean up of the generalised fragment. The idea is to correct obvious errors or omissions in the generalised trees. The clean up step is performed in the general spirit of the flat treebank design philosopy (i.e. at this stage we do not introduce Xbar assumptions). As a rough guideline for restructuring transformations, we use analyses attested elsewhere in the fragment. We illustrate this step by way of an example. Consider the very first NP in the (generalised) treebank fragment:

```
                                 np
          _______________________|_______________________
         /             |       |     |     |      |         \
        np            n0      n0    n0     p     np         tgp
   ______|_____        |       |     |     |    __|__        |
  /      |      \      |       |     |     |   /     \       |
 n0     n0     pnct  18-foot  jump  shot  with num    n0      v0
  |      |       |                              |      |       |
  |      |       |                              |      |       |
alvis  rogers gen-sym                          one  second  remaining
```

The example shows a missed PP post-modifier grouping at the tail of the NP. In addition we also relabel the initial NP daughter as a determiner phrase DETP:

```
                          np
          ________________|________________
         /             |       |     |        \
       detp           n0      n0    n0        pp
   ______|_____        |       |     |     _____|_____
  /      |      \      |       |     |    /           \
 n0     n0     pnct  18-foot  jump  shot  p            np
  |      |       |                        |      ______|______
  |      |       |                        |     /      |      \
alvis  rogers gen-sym                    with  num    n0      tgp
                                                |      |       |
                                                |      |       |
                                               one  second    v0
                                                               |
                                                               |
                                                           remaining
```

Notice that it would be possible to further restructure the initial DETP daughter into:

```
          detp
       ___|____
      /        \
     np        gen
   __|_         |
  /     \       ,
 n0      n0
  |       |
alvis  rogers
```

However, here we abstain from this restructuring as this analysis is not attested elsewhere in the treebank fragment.

Manual correction involved a total of 143 transformation events on the 100 trees in the generalised treebank fragment. The major transformation types can be classified in the following typology:

- missed groupings
- single node relabelling
- attachment restructuring
- surrounding punctuation

Due to space limitations here we cannot give a detailed discussion of the transformation types. The first two transformation types were illustrated in the example above. Attachment restructuring mainly concerns certain instances of pre- and post-modifier phrases while surrounding punctuation restructures opening and closing punctuation as daughters of the same mother node.

We then extract a grammar $G^{c,m}$ from the generalised and manually corrected treebank fragment. As expected, the extracted rule set is "cleaner" and "improved" compared to G^m. To give but a single illustration based on the NP example discussed above, G^m features a rule

```
np -> np n0 n0 n0 p np tgp
```

derived from the top level NP local tree while in $G^{c,m}$ this rule is replaced by

```
np -> detp n0 n0 n0 pp
```

$G^{c,m}$ features the same number of rule types as G^c, however, $G^{c,m}$ has more rule tokens (applications of rules) as evidenced by the labelled bracketing counts. Intuitively this means that the manually corrected treebank has more structure:

	# rule types	labelled bracketing
G	509	1318
G^c	331	1318
$G^{c,m}$	331	1422

In order to evaluate manual correction in terms of how well the extracted grammar performs under automatic feature-structure annotation, we manually annotate $G^{c,m}$ with feature-structure information. The results of the manual annotation provide a gold-standard for automatic annotation. We then develop a set of regular expression based annotation templates and apply these automatically to an unannotated version of $G^{c,m}$. We then compare the manually annotated grammar with the automatically annotated grammar in terms of precision and recall results and contrast the results obtained with the corresponding figures for G^c

	precision	recall
G^c	93.4 %	91.6 %
$G^{c,m}$	93.7 %	94.9 %

Precision and recall results for grammar $G^{c,m}$ extracted from the generalised and manually correct treebank fragment are improved. This is as expected. $G^{c,m}$ shows fewer quirks and "odd" rules compared to G^c. The strongest reflection of this is in the 3.3% increase in recall, meaning that automatic annotation generates more of the gold-standard target annotations compared to G^c.

7 Automatic Correction

Manual correction or clean up of a treebank resource is very time consuming, expensive, error prone and partial in that it risks missing appropriate corrections. In this section we apply a method for automatic treebank conversion, based on rule-based rewriting of trees encoded in a flat tree description language. The method is a special application of the automatic feature-structure annotation methodology developed in [Frank,00], yet applied to convert the structure of treebank trees. A similar approach to treebank conversion is presented in [Frank,01].

In the first step, trees are automatically translated into a set (a conjunction) of basic tree description predicates of a general tree description language. The tree description language features a number of basic and derived predicates:

Basic tree predicates (`A,B` node identifiers)

`arc(A,LA,B,LB)`	B daughter of A, with category labels LA,LB
`prec(A,B)`	immediate precedence A–B
`lex(A,LA,Lex)`	lexical node A with category label LA

Derived tree predicates (`A,B,..` node identifiers)

`dom(A,B)`	immediate dominance
`dom_x(A,C)`	dominance
`prec_x(A,C)`	precedence
`first_d(A,X)`	X first daughter of A
`last_d(A,Y)`	Y last daughter of A
`c_label(A,CA)`	CA functional label of A
...	possibly more ...

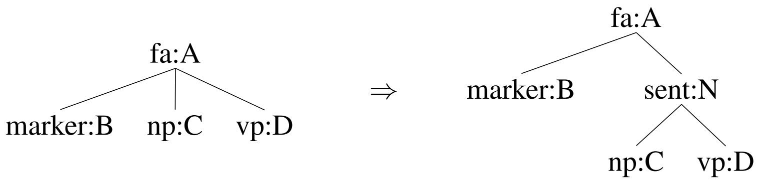
```
where:
dom(A,B) :- arc(A,_,B,_).
dom_x(A,C) :- dom(A,C) ∨ (dom(A,B) ∧ dom_x(B,C)).
prec_x(A,C) :- prec(A,C) ∨ (prec(A,B) ∧ prec_x(B,C)).
```

We then define parameterised tree transformation rules, using predicates (both basic and derived) of the tree description language. The rules apply to partial tree configurations, and rewrite (part of) the tree predicates, to derive a new set, and thus a new tree description. The new tree description is then reconverted to a standard tree representation. Tree transformation rules are of the form `ID :: CD >> TD` where `ID` is a rule identifier, `CD` a constraining tree description and `TD` a set of transformation operations.

Transformation rules are best explained by way of example. The following transformation rule identifies `np vp` sequences preceded by `marker`, preposition or prepositional phrase daughters (`subord_marker`) in `fa`, `fn` or `relcl` constituents (`sub_cl`), and "lowers" the `np vp` sequence below a newly created `sent` node within the original `fa`, `fn` or `relcl` constituent:

```
sub_cl :: sub_cl(A,CA),
```

```
    arc(A,CA,B,CB), subord_marker(A,CA,CB), prec(B,C),
    arc(A,CA,C,"np"), prec_x(C,D),
    arc(A,CA,D,"vp"), last_d(A,Y)
>>  new_dtr(A,N,"sent"),
    lower_subtree(A,C,Y,N,"sent").
```



We developed a set of 13 transformation rules. Together these transformation rules effect a total of 110 transformation events, compared to 143 manual transformations, with the manual transformation events being a near superset of the automatic transformation events (one of the automatic transformation events is not in the manual transformation events):

	AP100^{c,m}	AP100^{c,a}
# of transf. ev.	143	110

RuleType	# of transf. ev.
sub_cl	23
fn_to_sbar	3
fn_to_sent	26
existential	3
inv_sent	18
surrounding_commas1	11
surrounding_commas2	1
xtraposed_mods	2
detp_gen	8
adv_num_detp	8
unary_sent_np	1
pre_conj_conj	1
flat_coord	5

We evaluate grammar $G^{c,a}$ extracted from the automatically corrected treebank fragment AP100^{c,a} against the grammar $G^{c,m}$ extracted from the manually corrected treebank fragment AP100^{c,m} in terms of standard labelled bracketing (LB), labelled precision(LP) and labelled recall (LR) measures from parsing. Compared to the generalised treebank fragment AP100^{c}, both manual and automatic correction introduce more structure, manual correction more so than automatic correction:

	LB
AP100^{c}	1318
AP100^{c,m}	1422
AP100^{c,a}	1389

Labelled precision measures how many of the labelled bracketings in the automatically corrected treebank fragment are correct as defined by the manual correction. Labelled recall measures how many of the correct target labelled bracketings as defined by manual correction are generated by automatic correction:

	labelled precision	labelled recall
AP100^{c,a}	97.8 %	95.6 %

A more phenomenon oriented precision and recall measure is shown by the following figures, where we evaluate precision and recall of transformation events (rather than (labelled) bracketing divergences between manual and automatic corrections reflected in the LP and LR results in the table above). Here, 1 out of 110 automatic transformation events is not in the set of manual transformations; 34 manual transformation events are missed.

phenomenon oriented	precision	recall
AP100^{c,a}	99 %	76 %

8 Regular Expression Based Compaction

Grammars $G^{c,m}$ and $G^{c,a}$ look very much like flat, compiled out versions of LFG-82 style grammars [Kaplan and Bresnan,82]. These early LFG grammars allow regular expressions such as Kleene star "`*`" and optionality "`(..)`" in rule RHSs such as the following (here we only show the CFG part):

```
vp -> v (np) (np) (vp) (sent) pp*
```

Regular expressions in rule RHSs allow a compact representation of a multitude of CFG rules and this can greatly aid perspicuity for manual and automatic feature-structure annotation tasks. In our previous work we developed regular expression based feature-structure annotation principles designed to match large numbers of CFG rules extracted from treebank resources. In an inversion of this idea we could try to compact a treebank grammar using regular expressions in rule RHSs. Because of space limitations here we can only outline the basic ideas. We hope to report on this elsewhere. In our work to date we have explored two methodologies: the first is based on finite-state technology (FST) and automatic conversion into a Finite State Automaton (FSA); the second on manual conversion.

The basic idea in FST based automatic conversion is simple: for a given rule LHS, collect all RHSs attested in the grammar.

```
LHS > RHS1
LHS > RHS2
 ..     ..
LHS > RHSn
```

Each RHS constitutes a (trivial) regular expression. For a given LHS union all RHSs into a "disjunctive" regular expression:

```
[RHS1 | RHS2 | .. | RHSn]
```

Use FST tools to convert the regular expression into an automaton. Apply minimisation to the resulting automaton. Convert back to a regular expression. Automatic conversion along these lines (especially of large numbers of rule RHSs) does not always result in "readable" regular expressions. In fact it can be useful to manually split large sets of rule RHSs into "natural" classes prior to automatic conversion.

Manual conversion is based on the following idea. Suppose that a treebank attests the following rules:

```
np > det n0
np > n0 pp
```

these can be "compacted" into

```
np > (det) n0 (pp)
```

Note that the compaction is linguistically well motivated but that the language defined by the regular expression in the compaction is not the same as the language defined by the attested rule RHSs:

```
(det) n0 (pp) =/= det n0 | n0 pp
```

In fact, the compaction covers two unattested possibilities:

```
(det) n0 (pp) = det n0 | n0 pp | n0 | det n0 pp
```

We conducted a manual experiment which compacted $G^{c,m}$ into 20 rules with regular RHSs. We used our regular expression interpreter to verify that for each LHS in the grammar, the language defined by the regular expression based rule RHSs properly includes the language defined by the RHSs attested in the original grammar.

9 Xbar-based Annotation

In our final experiment we transform the first 50 parse-annotated sentences of the generalised and manually corrected version AP100^{c,m} of the treebank fragment into a "vanilla flavour" Xbar encoding AP50^{c,m,x}. Our choice of Xbar design principles is pragmatic and similar to other (computational) grammars in the literature: [Pollard and Sag, 94], [Butt et al,99]. It does not constitute deep syntactic theorising! Our design principles are:

1. three projection levels: X0, X1 and XP
2. X1 only present when required (as in adjunction)
3. complements are taken at X0 in one go (this means that we will have a number of flat non-binary branching rules)
4. adjuncts are taken at X1 recursively
5. only X2 (maximal projection) can act as complement/adjunct
6. no ϵ-productions

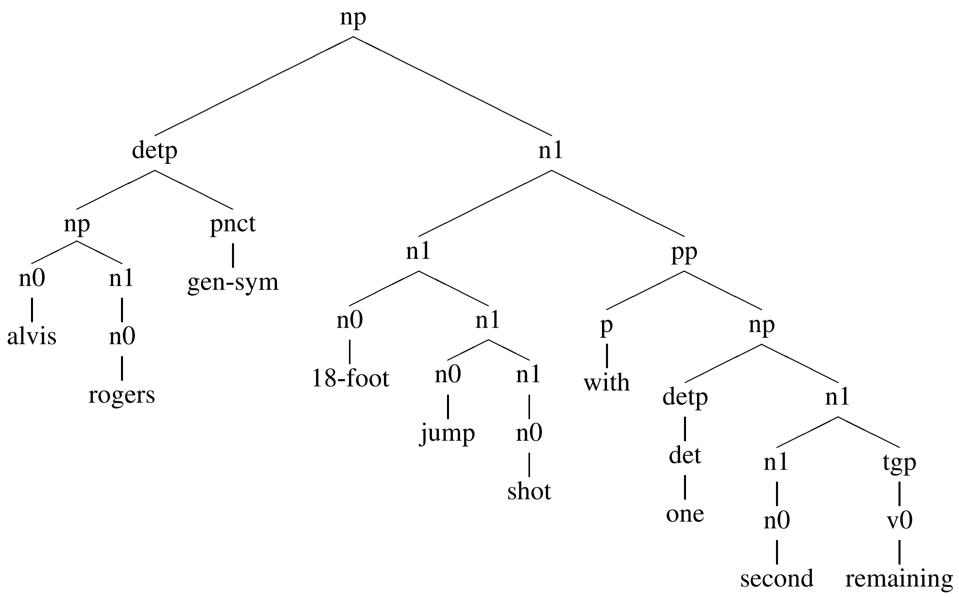
Principle 1 is pretty uncontroversial. Principle 2 (minimal representations: X1 only when needed) will have interesting consequences for the number of CFG rules extracted from the Xbar encoding. Principle 3 aids in functional annotation of subcategorisable grammatical functions (OBJ, OBJ2, XCOMP, COMP, . . .). Principles 4 and 5 are again fairly standard assumptions. Principle 4 has interesting consequences for probabilistic versions of CFGs extracted from such encodings. Finally, principle 6 is a reflection of the "surfacy" nature of the CFG component in LFG based grammar architectures.

Taken together principles 1 – 6 constitute fairly "mild" Xbar assumptions. Nevertheless, they are linguistic idealisations which motivate some of our "systematic" vanilla flavour Xbar violations:

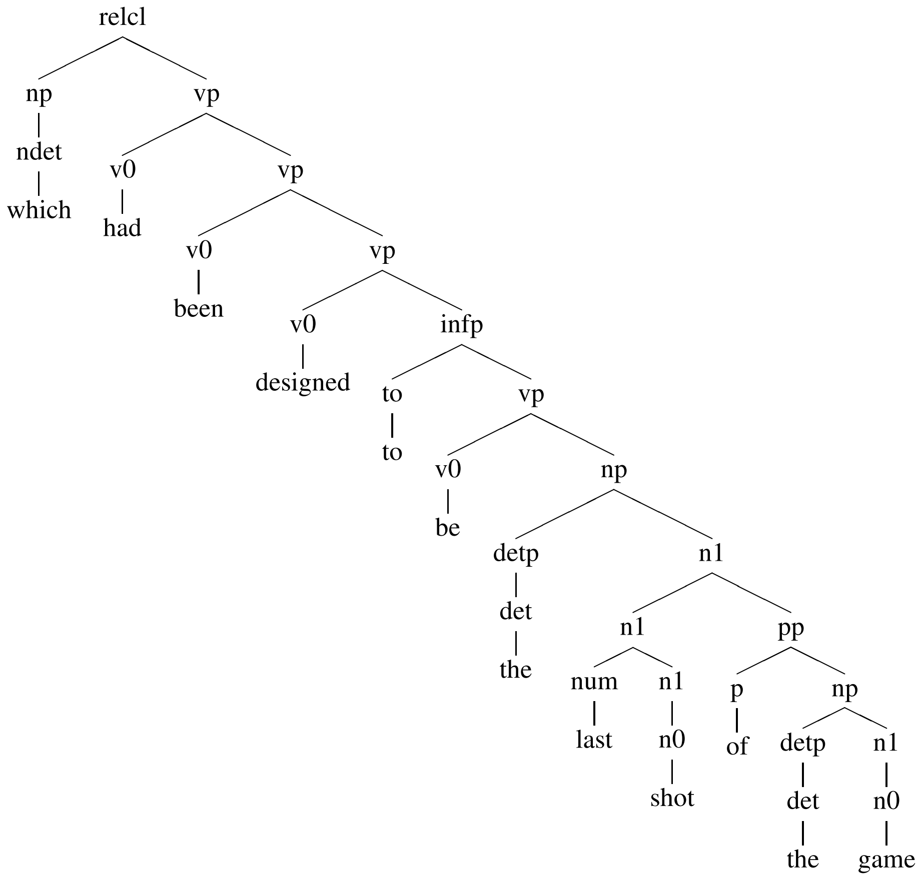
1. inside NP we allow for N0 as modifier of N1 and as a direct modifier daughter of NP (i.e. we do not always introduce N1 in N0 modifier configurations)
2. because of Xbar principle 3 we need to allow adjuncts between complements at X0
3. we keep some (traditional) treebank categories such as SENT, VP, TGP . . .
4. we do not provide Xbar accounts of coordination and punctuation

Xbar violation 1 is to account for our treatment of nominal compounds. Violation 2 (of principles 3 and 4) is motivated by the data, i.e. adjuncts occurring between complements. Violation 3 is less a violation than a partial commitment to Xbar-based generalisations. Violation 4 is again a pragmatic decision.

We illustrate both our Xbar design principles and violations with two examples. The first is a recoding of the initial NP from the first sentence in the treebank fragment:



The second is a complex relative clause:



Note that both trees are exclusively binary branching and that together with a few additional assumptions each local subtree identifies head, modifier and complement relations. The additional assumptions include, for example, that in general for English NPs the head projection follows the right branch (given our treatment of nominal compounds this does not follow from the vanilla flavour Xbar encoding). These additional assumptions together with our Xbar design principles and violations should guide automatic feature-structure annotations to a much greater extent than the original flat treebank grammar design.

In order to evaluate the Xbar recoding, we extract grammar $G^{c,m,x}$ from fragment AP50^{c,m,x}. We will give a few general observations on the extracted grammar. Compared to our other grammars the Xbar-based grammar features shorter rule RHSs, fewer rule types but substantially more rule tokens (local trees of depth one) per tree. In the table below we list the number of rule types extracted against increasing corpus size for the generalised and manually corrected fragment AP100^{c,m} and the Xbar based AP50^{c,m,x}:

# sentence	# rules AP50^{c,m,x}	# rules AP100^{c,m}	difference	difference %
05	47	37	+10	+27.03
10	66	56	+10	+17.86
15	78	78	0	0
20	94	102	- 8	-7.84
25	102	122	-20	-16.39
30	114	135	-21	-15.56
35	133	159	-26	-16.35
40	144	174	-30	-17.24
45	151	184	-33	-17.93
50	161	201	-40	-19.90

As expected, grammar $G^{c,m,x}$ extracted from AP50^{c,m,x} grows more slowly than grammar $G^{c,m}$ extracted from AP100^{c,m}. Xbar grammars features fewer, more general rules than flat treebank grammars with unabated rule accession rates. However, the effect is less pronounced than could be expected. The reasons for this are that

- according to our Xbar principle 3 which states that X1 is only featured if required, we get duplication of complementation rules at XP and X1 level depending on the presence or absence of modifiers:

  ```
  vp -> v0 np          v1 -> v0 np
  vp -> v0 np np       v1 -> v0 np np
  ```

  Ironically, more parsimonious parse trees yield a larger rule base.

- the fragments considered are too small to see the full effects of Xbar vs. treebank based rule accession rates.

In all our extracted grammars, the NP fragment constitutes the most varied and complicated subsection and thus poses the most difficult section for automatic feature-structure annotation. We isolate the NP section from $G^{c,m,x}$ and manually annotate it with feature-structure information to create a gold-standard against which to evaluate automatic annotation. We then develop regular expression based feature-structure annotation principles and automatically apply them to an unannotated version of the NP fragment from $G^{c,m,x}$. We then compare the results of automatic annotation against the gold standard in terms of precision and recall results.

As most of the CFG rules extracted from AP50^{c,m,x} are unary or binary branching (the only exceptions are rules accounting for coordination, punctuation and complementation), most of the regular expressions in the annotation principles are simple unions or contain underspecified category symbols:

```
n1:A > n1:B [adjp:C|advp:C|pp:C|recl:C|tgp:C|vp:C]
      @ [A=B, A:adjn $ C]
v1:A > v1:B __:C
      @ [A=B, A:adjn $ C]
```

The first annotation principle enumerates the possible `n1` post-modifiers in the grammar. The reason we cannot underspecify the modifier category symbol (as in the second `v1` annotation principle) is that the grammar extracted allows for binary branching

```
n1 > n1 n1
```

rules where the head projection follows the right branch. In general, annotation principles for $G^{c,m,x}$ are no longer underspecified (with the exception of principles dealing with the non-binary branching complementation, coordination and punctuation rules) as there is no need to determine subsequences in flat rule RHS for proper annotation as was the case for flat treebank rules. Automatic annotation should thus be much simpler. The precision and recall results for the NP fragment of $G^{c,m,x}$ and $G^{c,m}$ (for the first 50 sentences) support this expectation:

	precision	recall
$G^{c,m}$	93 %	94.4 %
$G^{c,m,x}$	100 %	100 %

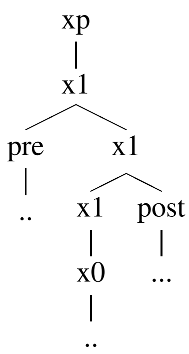
10 Conclusions and Further Work

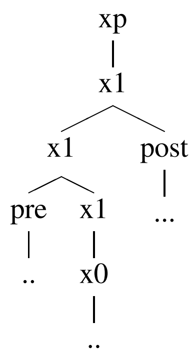
We have presented a number of experiments that move a flat treebank grammar fragment in stages to a vanilla flavour Xbar encoding. We have evaluated the resulting grammars with respect to how they perform in automatic feature-structure annotation tasks. Some of our experiments involved manual recoding of treebank resources. Manual recoding is extremely time consuming and error prone. We have investigated to what extent treebank grooming can be automated and have developed, applied and evaluated automatic treebank conversion procedures. We have sketched regular expression based compaction of rule RHSs extracted from treebank grammars. We hope to be able to expand on this in further work.

The results of our final Xbar encoding can be summarised as follows: vanilla flavour Xbar coding can be scaled but is extremely time and resource intensive. It provides a smaller, cleaner, and more general rule base which makes it easy to spot inconsistencies in the treebank (simply extract the grammar and search for odd rules). Xbar-based trees support (and in fact drive) automatic feature-structure annotation, a result that tallies well with work in theoretical linguistics [Bresnan,01]. Xbar-based trees require fewer and less partial automatic feature-structure annotation principles. However, Xbar design principles force subtle and time consuming scope/attachment decisions and currently do not support a notion of underspecification. This seriously limits its application in large scale treebanking tasks. Small and general rule bases are bought at the price of multiplying (often exploding) analysis possibilities with ensuing load under parsing and generation. What is more, Xbar-based rule sets are ill suited to probabilistic approaches to parsing (and generation) for at least the following two reasons:

First, Xbar-based CFG trees involve large numbers of rule tokens. Under standard PCFG assumptions, the probability contributed by each such rule token introduces an independence assumption into the computation of the probability of the tree and thus weakens the probability model.

Second, the standard Xbar assumption that pre- and post-modification (adjuncts) revolves around a recursive X1 level means that PCFG versions of such grammars in principle cannot distinguish high and low attachment possibilities in the presence of both a pre- and a post-modifier at a local X1 configuration





as the product of the individual rule probabilities involved in both trees are exactly the same.

Bibliography

(Bresnan,01) Joan Bresnan, *Lexical Functional Syntax*, Blackwell Publishers, Oxford, 2001.

(Butt et al,99) Miriam Butt, T.H. King and F. Second, *A grammar writer's cookbook*, Stanford, Calif. : CSLI Publications, 1999.

(Charniak,93) Eugene Charniak, *Statistical language learning*, Cambridge, Mass : MIT Press, 1993.

(Frank,01) Anette Frank, *Treebank Conversion. Converting the NEGRA Treebank to an LTAG Grammar*, in: *Proceedings of the Workshop on Multi-layer Corpus-based Analysis*, Iasi, Romania, 2001, pp. 29–43.

(Frank,00) Anette Frank, *Automatic F-Structure Annotation of Treebank Trees*, in M. Butt and T.H.King (Eds.), Proceedings of the LFG00 Conference, 2000, University of California, Berkeley, CSLI Online Publications, Stanford, CA, 2000, pp. 139-159,
http://www-csli.stanford.edu/publications/

(Frank et al,01) Anette Frank, Louisa Sadler, Josef van Genabith und Andy Way, *From Treebank Resources to LFG F-Structures. Automatic F-Structure Annotation of Treebank Trees and CFGs extracted from Treebanks*, to appear in: Abeille, A. (ed): *Treebanks. Building and using syntactically annotated corpora*, Kluwer Academic Publishers, The Netherlands, to appear, 23 pp.

(Hepple and van Genabith,00) Mark Hepple and Josef van Genabith, *Experiments in Structure-Preserving Grammar Compaction*, 1st Meeting on Speech Technology Transfer, Universidad de Sevilla and Universidad de Granada, Seville, Spain, Nov. 6th - 10th, 2000.

(Kaplan and Bresnan,82) Ron Kaplan and Joan Bresnan, *Lexical-Functional Grammar: a Formal System for Grammatical Representation*, in Bresnan, Joan (ed.), *The Mental representation of grammatical relations*, Cambridge, Mass : MIT Press, 1982.

(Krotov et al,98) Krotov, A., M. Hepple, R.Gauzauskas and Y. Wilks, Compacting the Penn Treebank Grammar, in *Proceedings of COLING/ACL'98*, Montreal, Canada, pp.699-703

(Leech and Garside,91) Leech, G. and R. Garside, *Running a Grammar Factory: on the Compilation of Parsed Corpora or Treebanks*, Mouton de Gruyter, Berlin, 1991, pp. 15-32.

(Pollard and Sag,94) Pollard, Carl and Sag, Ivan. *Head-driven phrase structure grammar*, Stanford : Center for the Study of Language and Information; Chicago; Univer, 1994

(Sadler, van Genabith, Way,00) Sadler,L., J. van Genabith and A. Way, *Automatic F-Structure Annotation from the AP Treebank*, in M. Butt and T.H.King (Eds.), Proceedings of the LFG00 Conference, 2000, University of California, Berkeley, CSLI ONline Publications, Stanford, CA, http://www-csli.stanford.edu/publications/

(van Genabith, Sadler and Way,99) Josef van Genabith, Louisa Sadler and Andy Way, *Structure Preserving CF-PSG Compaction, LFG and Treebanks*, Journees ATALA, Corpus annotes pour la syntaxe, Actes, (Proceedings ATALA Workshop - Treebanks), l'Universite Paris 7, France, 18-19 Juin 1999, pp.107-114, also http://talana.linguist.jussieu.fr/treebanks99/