AUTOMATIC F-STRUCTURE ANNOTATION FROM THE AP TREEBANK

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

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

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

Proceedings of the LFG00 Conference

University of California, Berkeley

Miriam Butt and Tracy Holloway King (Editors)

2000
CSLI Publications
http://www-csli.stanford.edu/publications/

Abstract

We present a method for automatically annotating treebank resources with functional structures. The method defines systematic patterns of correspondence between partial PS configurations and functional structures. These are applied to PS rules extracted from treebanks. The set of techniques which we have developed constitute a methodology for corpus-guided grammar development. Despite the widespread belief that treebank representations are not very useful in grammar development, we show that systematic patterns of c-structure to f-structure correspondence can be simply and successfully stated over such rules. The method is partial in that it requires manual correction of the annotated grammar rules.

1 Introduction

The present paper discusses a method for the automatic annotation of treebanks with functional structures. A companion paper (Frank 2000) presents an alternative method for the automatic annotation of corpus resources. These closely related, but interestingly different methods have developed through much collaborative interchange. We present them in two separate contributions to allow for more in-depth discussion and comparison. We first describe our method and then exemplify its application to a grammar of 330 rules derived from a fragment of the AP treebank. We give some results concerning precision and recall for this grammar.

Treebanks which encode higher-level functional structure information in addition to phrase structure information, are required as training resources for probabilistic unification grammars and data-driven parsing approaches, e.g. (Bod and Kaplan 1998). Manual construction of such treebanks is very labour and cost intensive. As an alternative, one could envisage the construction of new, or the scaling-up of existing, unification grammars which could then be used to analyze corpora. However, these approaches are equally labour and cost intensive. What is more, even if a large-coverage unification grammar is available, typically, for each sentence it would come up with hundreds or thousands of candidate analyses from which a highly trained expert has to select. Although proposals have been made for filtering and ranking parsing ambiguities (e.g. (Frank et al. 1998)), to date none is guaranteed to uniquely determine the best analysis. In order not to compromise the quality of the corpus under construction, a linguistic expert is required to find the best among a large number of candidate analyses.

As a partial response to this data problem, van Genabith et al. (1999a,b,c) introduce a method for bootstrapping the construction of grammars from treebank resources. Their basic idea is the following: take an existing treebank, read off the CF-PSG following (Charniak 1996), manually annotate it with f-structure annotations, provide macros for the lexical entries and then "reparse" the treebanked trees simply following the original c-structure annotations. During this reparsing process, the f-structure annotations are resolved, and an f-structure is produced. The process is deterministic if the annotations are, and to a large extent costly manual inspection of candidate analyses is avoided. The method successfully allows the creation of grammar resources but still involves one labour intensive manual component, namely annotation of the grammar rules with functional information. Much recent work in LFG, however, has shown that the c-structure f-structure correspondence for a configurational, generally endocentric language such as English, is largely predictable from a small set of mapping principles (King 1995, Kroeger 1995, Bresnan 2000). In the approach of Bresnan (2000) and colleagues, the mapping principles assume a highly articulated set of X'-schemata involving both functional and lexical projections in a configurational language such as English. A similar, but largely implicit, assumption about the predictability of the c- to f-structure mapping is also present in the earlier work in LFG (Kaplan and Bresnan 1982) where it turns up essentially as constraints on pairings of categories and grammatical functions (e.g. COMP is only appropriate for S, only NPs/DPs are OBJs and so forth). In general, the correspondence between c-structure and f-structure follows from

linguistically determined principles which are partly universal, and partly language specific (Bresnan 2000), (Dalrymple 1999).

In the light of this, an obvious strategy to pursue is to implement a set of principles to automatically provide f-structure annotations of CFG rules derived from treebank representations, eliminating the manual step in the previous method. As a side effect, this can be expected to cast light on the soundness, accuracy and appropriacy of the linguists' generalisations: that is, the automatic procedure as applied to a large ruleset derived from a treebank, can serve as a potentially interesting testbed for the linguistic principles.

This paper substantially extends the research in van Genabith et al. (1999a,b,c) by showing how f-structure annotations of grammar rules extracted from treebanks may (to a large extent) be automated. The basic idea is very simple. We read off a CFG treebank grammar, using the first 100 trees of the AP treebank (Leech and Garside 1991). Systematic correspondences between elements in the c-structure domain and elements in the f-structure domain are then defined in general annotation templates. A corrected/completed version of this grammar is then used to induce f-structure assignments for PS trees from the treebank following the reparsing method of van Genabith et al. (1999a,b,c). The method is partial in that it requires manual inspection and correction of the output produced by the automatic annotation process. The method results in a set of annotated rules for real text.

The potential benefits of automation are considerable: substantial reduction in development effort, hence savings in time and cost for treebank annotation and grammar development; the ability to tackle larger fragments in a shorter time, a considerable amount of flexibility for switching between different treebank annotation schemes, and a natural approach to robustness. The method we present may be viewed as a corpus-guided grammar development methodology.

The paper is structured as follows. In Section 2 we introduce the formalism for writing annotation templates. In Section 3 we discuss in some detail the NP fragment of our grammar and present a number of the templates involved. Section 4 presents the design of the automatic annotation experiment and evaluates the results obtained. Finally we conclude and outline further work.

2 Automatic f-structure annotation of CF Rules

Treebank grammars (CFGs extracted from treebanks) are very large and grow with the size of the treebank (Charniak 1996), (Krotov et al. 1998). They feature flat rules, many of which share and/or repeat significant portions of their RHSs. This causes several problems for manual annotation approaches such as the one described in van Genabith et al. (1999a,b,c). Annotation is labour intensive and repetitive, because of the sheer size and similarity of the rules, and annotation of rules on a one by one basis means that generalisations known to the annotator are simply not expressed. Of course, if the cardinality of the ruleset continues to grow with the size of the treebank, so too will the manual annotation task.

In LFG the correspondence between functional and constituent structure is partly defined in terms of annotations associated with c-structure nodes. Annotation follows universal and language specific principles. We can define principles as involving *partial* phrase structure configurations and apply them to all CFG rules that meet the relevant *partial* configuration. To give a simple example: a head principle assigns $\uparrow = \downarrow$ to the X daughter in all XP $\rightarrow$ ...X... configurations, irrespective of the surrounding categorial context. Such annotation principles capture generalisations, which can be used to *automatically* annotate PS configurations with functional structures in a highly general and economical way.

2.1 Feature Description Templates

In our approach to automatic annotation of c-structure rules, the linguist states generalisations over local sub-trees in the form of feature description templates, defined as follows:

```
fd(Rule,Constraints,FDescr)
```

In a template, the `Rule` is a (possibly partial) description of a treebank rule, while `Constraints` is a list of categorial and configurational (that is, c-structure) constraints and `FDesrc` a list of functional annotations induced. The intended interpretation is that every rule that satisfies `Constraints` is annotated with `FDescr`. Rules can match, satisfy and receive annotations from multiple `fd/3` templates. The interpretation of templates is order independent: every template whose description is satisfied is applied to a given rule.

The constraint interpreter supports a wide range of different types of expression including list constraints, restricted regular expressions, macros and user defined constraints. At this exploratory stage, we have not tailored the constraint interpreter to enforce any particular style of constraint specification by the linguist, and thus we provide a number of constraint predicates which are perhaps unlikely to be linguistically motivated. As we shall see below, because we are dealing with the rather flat representations of treebank entries, we rely on constraints over linearity rather than hierarchical structure in a number of cases.

To give a flavour of the template description language, the following functions are included in the list processing constraints:

(1) `one(Dtrs,O)`: returns the only item `O` of a list `Dtrs`. It fails if `Dtrs = []`, if `O` is not an item in `Dtrs` or if `O` occurs more than just once in `Dtrs`. `one/2` thus has the meaning of exactly once. `Dtrs` may contain regular expressions. This is a deterministic predicate.

`first(Dtrs,F)`: returns the first item `F` of a list `Dtrs`. It fails if `Dtrs = []`. `Dtrs` may contain regular expressions. Deterministic predicate. `second(Dtrs,S)`, and so on, are similarly defined.

`last(Dtrs,L)`: returns the last item `L` of a list `Dtrs`. It fails if `Dtrs = []`. `Dtrs` may contain regular expressions. Deterministic predicate.

`eq(List1,List2)`: succeeds if `List1` and `List2` are equal. `List1` and `List2` may contain regular expressions. Nondeterministic predicate (lists containing regular expressions cam be the same in more than one way).

`leftof(D,Dtrs,LDtrs)`: `LDtrs` is the list to the left of item `D` in the list `Dtrs`. `Dtrs` and `LDtrs` may contain regular expressions. Nondeterministic predicate. `LDtrs = []` if `D` first item in `Dtrs`. A predicate `rightof` is similarly defined.

`element(D,Dtrs)`: `D` is an item in the list `Dtrs`. Processing is left-to-right. `Dtrs` may contain regular expressions. Nondeterministic predicate. A deterministic element predicate is also defined.

`prefix(Pre,List)`: succeeds if the list `Pre` is a prefix of the list `List`. `Pre` and `List` may contain regular expressions. `[]` is a prefix of every list. Also, every list is a prefix of itself. Deterministic predicate. A `suffix` relation is similarly defined.

`sequence(Dtrs,Left,Sequence,Right)` : succeeds if the list `Sequence` is a subsequence (i.e. an infix) of the list `Dtrs`. `Left` and `Right` are the remainder lists to the left and right, respectively, of `Sequence`. `Left`, `Right`, `Sequence` and `Dtrs` may contain regular expressions. `[]` is an infix of every list. Also, every list is an infix of itself. Nondeterministic predicate.

As is evident from the above, we define a number of constraints with the very flat structures of treebank representations in mind. Often the linguist essentially picks out in a specific template a substring of the RHS categories which would have corresponded to a distinct non-terminal node in a more hierarchical representation, effectively creating a 'virtual' tree for the purposes of the mapping.

We provide a limited number of basic regular expressions, including a bounded form of the Kleene operator and an `opt` predicate which specifies that its argument list is optional. Because the formalism is simple and flexible, it is easy to extend: user defined Prolog predicates may be included in the list of structural constraints. For example, the predicate `unary(M,D,FD)` was defined to express the generalisation that for any unary branching grammar rule, the mother's f-structure is the same as the daughter's f-structure.

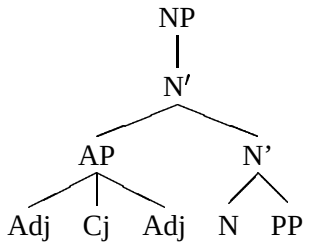
(2)
```
unary(M,D,[FM === FD]) :-
    M =.. [_Cat,FM],
    D =.. [_Cat,FD].
```

The constraint can then be used in a template as follows:

```
fd(rule(M,[D]),[unary(M,D,FDescr)],FDescr).
```

The formalism also contains a number of macro definitions. The most significant of these, the `conj/5` conjunction macro, describes and applies to regular patterns of daughters in RHSs of rule descriptions which together form a coordinate structure. Because the treebank representations are unduly flat, coordinate structures are very often not represented as separate subtrees, so the macro must pick out the relevant substring of daughters. For example, instead of the tree (3) we have the rule (4) from the treebank grammar.

(3)



(4)
```
rule(np(_), [adj(_),conj(_),adj(_),n0(_),pp(_)]).
```

The formalism also contains a number of interpreter control statements. The `oneof/1` control statement is satisfied as soon as one of the members of its argument list has been satisfied. The `all/4` constraint ensures that all constraints in its constraint list are applied as often as possible to a rule description and that the resulting annotations are collected. Consider an S rule which contains a number of PP daughters, or an NP rule which contains a number of AP daughters. The linguistic generalisations that one might want to state might be "map a PP daughter of S to ADJUNCT" or "map an AP daughter of NP to ADJUNCT". These are true of all such instances. The linguist uses the `all/4` interpreter control statement, `all(Rule,Constraints,FD,AllFD)`, to express this generalisation.[1]

(5)
```
all(rule(s(S),Dtrs),[member(pp(PP),Dtrs)],[S:s_adjunct:1 === PP],FD)
```

The first argument of `all(Rule,Constrs,FD,AllFD)` is a rule description `Rule`, the second argument a list of constraints `Constrs`, the third argument is the f-description `FD` induced each time the list of

[1] We abstract away here, for the purposes of exemplification, from the manner in which the adjunct set is modelled.

constraints is satisfied and finally the fourth argument `AllFD` collects and returns all those f-descriptions. The interpretation of `all/4` is that each way of satisfying `Constrs` in `Rule` generating a `FD` is collected in `AllFD`. The `all/4` constraint is included in the list of constraints in a `fd/3` template description:

(6)
```
fd(rule(s(S),Dtrs),
   [all(rule(s(S),Dtrs),[member(pp(PP),Dtrs)],[S:s_adjunct:1 === PP],FD)],
   FD).
```

The compilation of `fd/3` feature annotation templates over rule sets is provided in terms of a small and convenient top level program called `aa.pl` (Automatic Annotation). `aa.pl` loads the interpreter, the grammar file and the template file, and the top level predicate `aa` compiles out `Templates` over `Rules` and displays the result.

The processing of `fd/3` templates is order independent. Unless specified otherwise (e.g. by the `all/4` constraint) a template will apply only once to a rule description and can only apply again on backtracking. All templates that match a given rule will apply to that rule.[2]

3 The NP Grammar and Templates

In the previous section, we introduced the formalism for writing annotation templates. To give a flavour of what is involved, in this section we will present several aspects of the NP grammar. We show that this approach permits the linguist to state simple generalisations and translate them straightforwardly into templates. In our work to date we have developed templates for the entire grammar of 330 rules derived from the treebank. However, we have chosen to concentrate on one section of the grammar for expository purposes.

The NP fragment constitutes the largest and most complex set of phrase structure rules induced for a single non-terminal category from our set of sentences. Because of its size and complexity, and because the issues which it raises give a good feel for what is involved in our approach to automatic annotation, we limit discussion to this fragment. The grammar fragment contains 142 rules, for which we have written 29 templates. 23 categories are attested within NP, a very high proportion of the overall number of categories in the grammar, which is 41 (29 lexical and 12 non-terminal categories)[3].

(7) Categories found within NP

det	ndet	adj	adjp	dadj
n0	np	num	title	posspron
pron	pnct	conj	relcl	pp
p	ntadv	adv	v0	vp
fn	tgp	infp		

3.1 Compaction and Supercategories

In earlier work (van Genabith et al. 1999c) we found that the rich set of tags used in the AP treebank provided much useful f-structure information which could be simply re-expressed in a set of lexical macros.

[2] A manual with detailed descriptions of the interpreter and the constraints supported is available at `http://www.compapp.dcu.ie/~away/Treebank/treebank.html`.

[3] The category set that we are dealing with is derived from the original AP tagset by a process of compaction, which we describe in the following section.

The distinctions introduced by the AP tagset, in common with other tagsets, are extremely fine-grained because all sorts of subcategorial distinctions are expressed by means of the monadic category labels. In very many cases, these subcategorial distinctions are ones which would be expressed by means of grammatical features at f-structure in LFG (distinctions such as number, verbform, and so on). Since this information is recaptured by means of these lexical macros, they hypothesized that it would be helpful to abstract away from the specificities of the particular set of tags used in their database of sentences in favour of a smaller set of "supertags" in order to develop a stand-alone resource. This can be viewed as plugging holes in the grammar, for it permits a more general grammar to be derived from that which would otherwise be read off from the treebank entries. Therefore van Genabith et al. (1999b) introduce a structure-preserving grammar compaction method which first uses lexical macros to associate f-structure constraints with the words in the tree and then, having specified a mapping between tags and "supertags" (generalisations over tags), uses the latter to reparse the treebank entries and compile a "generalised" CFG from the tree using the method of (Charniak 1996).

The work described here investigates the automatic association of f-structure contstraints with the rules of the CFG by means of annotation templates. We have found that the categorial compaction described in (van Genabith et al. 1999b) has provided an excellent basis for automatic f-annotation: many of the distinctions preserved in the reduced (generalised) tagset are precisely those which we require to guide automatic annotation. For example, in the nominal domain, the large number of distinctions made between nominal elements on the basis of morphosyntactic class membership are eliminated, but the distinction of nouns with adverbial function is maintained. We make use of such information to directly guide the f-structure annotation process. Likewise, the AP tagset makes a series of distinctions within the verbal/sentential system which, suitably generalised over, are useful in the same way. As an example, we assign supertags over sets of AP tags as indicated below:

(8)

Supertag	AP Tag	Description
FA	Fa	Adverbial clause
	Fa&	First conjunct of an adverbial clause
	Fa+	Second conjunct of an adverbial clause
FN	Fn	Noun clause
	Fn&	First conjunct of a noun clause
	Fn+	Second conjunct of a noun clause
RELCL	Fr	Relative clause
	Fr&	First conjunct of a relative clause
	Fr+	Second conjunct of a relative clause
INFP	Ti	to + infinitive clause
	Ti&	First conjunct of a to + infinitive clause
	Ti+	Second conjunct of a to + infinitive clause

The following, exceptionless generalisations can be stated about these derived categories.

(9) An FN within NP is a COMP in the NP's f-structure

```
fd(rule(np(NP),Dtrs),
   [ all(rule(np(NP),Dtrs),
         [ element(fn(B),Dtrs) ],
         [ NP:comp === B ],
         AllConstr ) ],
   AllConstr ).
```

(10) An infinitival VP within NP is an XCOMP in the NP's f-structure

```
fd(rule(np(NP),Dtrs),
   [ all(rule(np(NP),Dtrs),
         [ element(infp(B),Dtrs) ],
         [ NP:xcomp === B ],
         AllConstr ) ],
   AllConstr ).
```

(11) A RELCL within NP is a RELMOD in the NP's f-structure

```
fd(rule(np(NP),Dtrs),
   [ all(rule(np(NP),Dtrs),
         [ element(relcl(B),Dtrs) ],
         [ NP:relmod === B ],
         AllConstr ) ],
   AllConstr ).
```

Of course, not all constituents which map to the f-structure function RELMOD *are* represented as **relcl** in the treebank entries. The sample of 100 sentences contains a number of cases of reduced relative clauses, which are associated with the (super-)category **vp** in our collapsed tagset. Given the distinctions made in the verbal supertag set, the following generalisation may be made about the occurrence of the (super-)tag **vp** within the noun phrase:

(12)
```
fd(rule(np(NP),Dtrs),
   [ all(rule(np(NP),Dtrs),
         [ element(vp(B),Dtrs) ],
         [ NP:relmod === B ],
         AllConstr ) ],
   AllConstr ).
```

Since the AP tagset encodes adverbial function, the supertag **ntadvp** (for nominal temporal adverbial) can be straightforwardly related to a specific function:

(13) An NTADVP maps to an NP_ADJUNCT in the mother's f-structure

```
fd(rule(np(NP),Dtrs),
   [ all(rule(np(NP),Dtrs),
         [ element(ntadvp(B),Dtrs) ],
         [ NP:np_adjunct:1 === B ],
         AllConstr ) ],
   AllConstr ).
```

We note at this point that our constraint language can also be implemented in terms of regular expressions. For instance, the template in (13) can be rewritten as:

(14)
```
np:NP > * ntadvp:B *
            @ [NP:np_adjunct:1=B]
```

We shall not give further examples of the regular expression variants on our templates here, but it is appropriate to note that we have subsequently implemented a basic regular expression based annotatator and our work henceforth will use this simpler, cleaner looking format[4]. Nevertheless, we stress that all experiments and results documented here were performed using the original `fd/3` version of the templates. While we do not see any reason why similar results cannot be obtained with the regular expression format of our templates, we have not tested this definitively at this stage.

The original AP tagset contains more than 20 pronominal tags, which we collapse to two supertags: **posspron** for possessive pronouns, and **pron** for all other pronouns. Again, the c-structure to f-structure mapping templates are simple to write for these categories[5]:

(15)
```
fd(rule(np(NP),Dtrs),
   [ all(rule(np(NP),Dtrs),
         [ element(posspron(B),Dtrs) ],
         [ NP:poss === B ],
         AllConstr ) ],
   AllConstr ).
```

(16)
```
fd(rule(np(NP),Dtrs),
   [ all(rule(np(NP),Dtrs),
         [ element(pron(B),Dtrs) ],
         [ NP === B ],
         AllConstr ) ],
   AllConstr ).
```

To take a more complicated example, the AP tagset distinguishes the following subtypes of Adjectives: **ja jb jj da da2 dar dat**. All the "j" tags are adjectives, either predicative, central and attributive. The "d" adjectives are "after determiners" such as "former, such, few, several...". The "j" adjectives are attributive modifiers within NP, and under our treatment, correspond to NP_ADJUNCT and HEADMOD grammatical functions,[6] while the "d" adjectives map to SPEC or may serve as the head of NP in the absence of a nominal element. In our collapsed tagset, all the "d" adjectives are treated as **dadj** and all the "j" adjectives as **adj**: this distinction, which is a simple generalisation of the categorial distinctions made in the treebank tags, corresponds to a difference in grammatical functional possibilities for these subtypes of adjectives. For **dadj**, the generalisation that we wish to state is that it maps to SPEC if there is a nominal f-head, otherwise to f-head.

(17)
```
fd(rule(np(NP),Dtrs),
   [element(dadj(DA),Dtrs),
    oneof([ element(n0(_),Dtrs),          %% as soon as one of them
            element(num(_),Dtrs),         %% is satis-
fied constraint
            element(np(_),Dtrs)  ]) ],    %% fires exactly once
   [NP:spec === DA]).
```

[4]Thanks to an anonymous reviewer for pointing this out.

[5]Of course, given the flatness of the trees, a **posspron** might denote a POSS function, but not necessarily the POSS of the f-structure of its mother node (it might be more deeply embedded in the f-structure). However, in our fragment this is not attested and we can make do with the simple generalisation in (15).

[6]We discuss this distinction at greater length below.

(18)
```
fd(rule(np(NP),Dtrs),
    [element(dadj(DA),Dtrs),
     not(element(num(_),Dtrs)),
     not(element(n0(_),Dtrs)),
     not(element(np(_),Dtrs))],
    [NP === DA]).
```

This pair of templates is essentially equivalent to annotating a **dadj** node with a disjunction ($\downarrow \in (\uparrow \mathrm{ADJ})) \vee (\downarrow = \uparrow)$. We recognise the ability of these adjectives to stand on their own as the head of NP by permitting the category **dadj** to serve as the head (when no other potential head is present) rather than by reassigning them to a nominal or determiner category.

The other subclass of adjectives serve as nominal modifiers within NP. The LFG treatment of attributive adjectives is as members of the set-valued feature ADJUNCT (here NP_ADJUNCT). This is appropriate for iterative uses of adjectives which separately restrict the interpretation of the head noun. However, our corpus contains a significant number of cases in which an adjective may appear on the left periphery (and part of) what is essentially a complex (internally-modified) nominal head, as in *jump shot*, *national guard troops*, *wide area telephone service*. For the latter cases we have used the additional grammatical function HEADMOD: the prototypical use of this function is in cases of noun-noun compounding, which abound in our small corpus extract. We shall have more to say about the HEADMOD function when we discuss NN compounds below.

Treating adjectives as potentially mapping to HEADMOD as well as to NP_ADJUNCT leads to the following template information for **adj**: adjectives next to nominal heads are either NP_ADJUNCTs or HEADMODs, other adjectives are NP_ADJUNCTs.

(19)
```
fd(rule(np(NP),Dtrs),
    [ all( rule(np(NP),Dtrs),
           [ sequence(Dtrs,[adj(A),XP]) ,
             element(XP,[n0(N0),np(N0),num(N0)]) ],
           [ (NP:np_adjunct:1 === A ; N0:headmod === A) ],
           AllConstr )
    ],
    AllConstr ).
```

(20)
```
fd(rule(np(NP),Dtrs),
    [ all( rule(np(NP),Dtrs),
           [ sequence(Dtrs,[adj(A),XP]) ,
             not(XP = n0(_)),
             not(XP = np(_)),
             not(XP = num(_)) ],
           [ NP:np_adjunct:1 === A ],
           AllConstr )
    ],
    AllConstr ).
```

Since only single, uncomplemented and unmodifed adjectives can be used as part of these sorts of structures, the template for AP is simple to state: it maps to the nominal adjunct function.

(21)
```
fd(rule(np(NP),Dtrs),
   [ all(rule(np(NP),Dtrs),
         [ element(adjp(B),Dtrs) ],
         [ NP:np_adjunct:1 === B ],
         AllConstr ) ],
   AllConstr ).
```

3.2 PP Dependants

Distinguishing OBL from ADJUNCTs is a notoriously difficult problem, especially within NPs, where the PP dependants are largely optional. One possible tack would be to treat all PP dependants of nominals as ADJUNCTs (this approach is adopted in (Butt et al. 1999)), that is, to treat all optional arguments of nominal heads as syntactic modifiers (some of which will be semantic arguments) rather than syntactic arguments. On this view, our annotation templates would simply need to state the generalisation that the category `pp` maps to the ADJUNCT function. However, inspection of our set of sentences suggests that while the second of two PPs is always an ADJUNCT, a PP adjacent to the nominal head may be either an OBL or an ADJUNCT. Although it is claimed that OBLIQUE and ADJUNCT PPs can reorder rather freely, the strings in the template reflect the ordering which would be imposed by the X'-schemata. The following templates, therefore, introduce a measure of disjunction into the annotation process:

(22)
```
fd(rule(np(NP),Dtrs),
   [ all(rule(np(NP),Dtrs),
         [ sequence(Dtrs,[pp(_),pp(C)]) ],
         [ NP:np_adjunct:1 === C ],
         AllConstr ) ],
   AllConstr ).
```

(23)
```
fd(rule(np(NP),Dtrs),
   [ all(rule(np(NP),Dtrs),
         [ sequence(Dtrs,[XP,pp(C)]) ,
           element(XP,[n0(_),np(_),num(_)]) ],
         [ (NP:np_adjunct:1 === C ; NP:obl === C) ],
         AllConstr ) ],
   AllConstr ).
```

3.3 Head Modifier Structures

The treebank representations of NPs are very flat and often quite complex - the following are representative.

(24)
```
rule(np(A), [det(B),adj(C),adj(D),n0(E),n0(F),adjp(G)]).
rule(np(A), [det(B),adj(C),n0(D),n0(E),relcl(F)]).
rule(np(A), [det(B),adj(C),n0(D),n0(E)]).
rule(np(A), [det(B),adj(C),n0(D),ntadvp(E),relcl(F)]).
rule(np(A), [det(B),adj(C),n0(D),pnct(E),vp(F)]).
rule(np(A), [det(B),adj(C),n0(D),pp(E)]).
rule(np(A), [det(B),dadj(C),n0(D),n0(E),n0(F),relcl(G)]).
```

```
rule(np(A), [n0(B),n0(C),n0(D),n0(E),vp(F)]).
rule(np(A), [n0(B),n0(C),n0(D),n0(E)]).
rule(np(A), [n0(B),n0(C),n0(D),np(E)]).
rule(np(A), [n0(B),n0(C),n0(D),ntadv(E),pp(F)]).
rule(np(A), [n0(B),n0(C),n0(D)]).
rule(np(A), [n0(B),n0(C),np(D),pnct(E),np(F)]).
rule(np(A), [n0(B),n0(C),np(D)]).
rule(np(A), [n0(B),n0(C),pnct(D),np(E)]).
rule(np(A), [n0(B),n0(C),pnct(D),relcl(E)]).
```

Given the remarkable paucity of internal structure here, a major issue is determining what category is the head of NP; that is, what category is to be annotated (↑= ↓). A striking feature of many of the NP rules is that they contain strings of nominal categories. In such cases, these flat strings of nominal categories behave in an essentially right-headed fashion. The elements **n0**, **num** and **np** typically serve as the head and the rightmost such element present (provided it is not preceded by **pnct**, which marks an appositional structure), is the head of the NP. This generalisation can be stated as follows:

(25)
```
fd(rule(np(NP),Dtrs),
   [ element(N,[n0(F),np(F),num(F)]),     % what exactly is N?
     sequence(Dtrs,Left,[N],Right),
     not(element(n0(_),Right)),           % check whether
     not(element(np(_),Right)),           % N is final
     not(element(num(_),Right)),
     not(last(Left,conj(_))),             % don't apply in
                                          % conj context
     not(last(Left,pnct(_))) ],           % nor if pre-
ceded by pnct  <==
   [NP === F]).
```

As can be seen, this template must take into account the complicating factor of coordination, and in particular, the flat representation of coordination, which we discuss in the following section. The fragment of grammar in (24) and the template (25) illustrate nicely a feature of treebank representations, namely their extremely flat representations. The template above must search for the rightmost category appropriate to serve as head, from a sequence of categories.

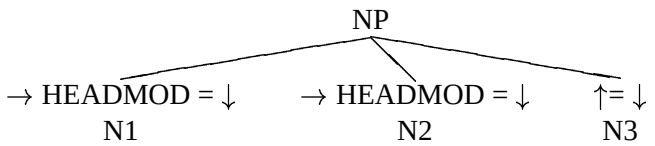
The overly flat treebank representations are very problematic when it comes to determining the correct head-modifer relationships within the noun phrase. One possibility is to treat all pre-head nominal (and adjectival) elements as direct modifiers of the final nominal head. It would be trivial to then define the appropriate annotation template mapping all such prehead nominal modifiers into a set-valued ADJUNCT f-structure. This would entail, for example, treating *law enforcement officer* as a head *officer* modified by a set of adjuncts { *law*, *enforcement* }. This is essentially the approach adopted in the LFG grammars of the PARGRAM project described in (Butt et al. 1999). The difficulty with this is that a flat representation as ADJUNCTS at f-structure would fail to encode the semantic modification relations which hold within these pre-head modifiers (although, of course, it is possible to keep trace of at least linear position in the string of modifiers by judicious indexing of the elements in the ADJUNCTS set).

These structures are extremely common in our fragment: for example, there are 52 rules containing a total of 72 simple **n0 n0** sequences in which nominal elements modify nominal structures to their right. We treat these as head-modifer structures, introducing a new (single-valued) grammatical function HEADMOD. Strings such as *guard helicopters*, *smoke inhalation*, *hospital spokesman*, *law enforcement officers*, and

many others in our sample, may be viewed as syntactic structures (each word is associated with a separate terminal category in the treebank representations) which are built according to "morphological" principles (number is marked on the final element, the structures are head final), in which each element modifies the f-structure of the element to its right.

This approach is also problematic, however. A nominal is taken to modify the f-structure projected from its nominal sister, as shown schematically in (26), where (→) denotes the f-structure of the immediately adjacent right sister, with the corresponding schematic f-structures in (27).

(26)



(27)
$$\begin{bmatrix} \text{PRED} & \text{'N3'} \\ \text{HEADMOD} & \begin{bmatrix} \text{PRED} & \text{'N2'} \\ \text{HEADMOD} & \begin{bmatrix} \text{PRED} & \text{'N1'} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

It is clear, however, that such "left-branching" structures are not always correct for the strings in our corpus, and that in some cases a "right-branching" f-structure, with a complex head (as shown in (28) would be more correct[7]. These sorts of structures, with what amounts to complex PREDs, are not permissible in LFG.

(28)
$$\begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{PRED} & \text{'N3'} \\ \text{HEADMOD} & \begin{bmatrix} \text{PRED} & \text{'N2'} \end{bmatrix} \end{bmatrix} \\ \text{HEADMOD} & \begin{bmatrix} \text{PRED} & \text{'N1'} \end{bmatrix} \end{bmatrix}$$

For the moment, however, the template simply picks out sequences of **n0** categories and adds the f-structure constraint that the first is the HEADMOD of the second. Note that the template cannot, of course, equate the f-structure of the mother with the rightmost category in the pair, since the f-structure of this category may itself be a HEADMOD within a containing f-structure: picking out the head of NP is performed by the head template given in (25) above.

(29)
```
fd(rule(np(_),Rhs),
    [ all( rule(np(_),Rhs),
           [ sequence(Rhs,[n0(N01),n0(N02)]) ],
           [ N02:headmod === N01 ],
           AllConstr ) ],
    AllConstr).
```

[7]Nevertheless, it must be stated that such NPs in our grammar are all treated as 'left-branching' structures. As we point out, this means that in some cases we provide the wrong treatment for such phenomena. The results given later in the paper were performed on this 'faulty' set of NPs. We are confident that reinterpreting these N-N compounds correctly will not prevent us from achieving equally good figures for precision and recall, but we have yet to rewrite the grammars and templates as desired, so that this must remain as speculation at this stage.

What else, apart from **n0**, maps to the headmod function? Members of the category **num**, like **adj** may be either ADJUNCTs or HEADMOD: the template is shown in (30). We extend the same treatment to titles, as in (31).

(30)
```
fd(rule(np(NP),Dtrs),
   [ all(rule(np(NP),Dtrs),
         [ sequence(Dtrs,[num(Num),n0(N0)]) ],
         [ ( N0:headmod === Num ; N0:np_adjunct:1 === Num) ],
         AllConstr ) ],
   AllConstr ).
```

(31)
```
fd(rule(np(NP),Dtrs),
   [ all(rule(np(NP),Dtrs),
         [ sequence(Dtrs,[title(T),n0(N0)]) ],
         [ N0:headmod === T ],
         AllConstr ) ],
   AllConstr ).
```

3.4 Coordination and Flat Trees

The approach to constituent coordination in LFG treats the conjuncts as a set at f-structure, with the conjunction contributing a value directly to the semantic structure. Our formalism does not currently support set values, and we model constituent coordination by treating the conjunction as a predicate taking a CONJ argument which itself takes any number of indexed arguments. The treatment of the conjuncts themselves then closely resembles our treatment of the set valued feature ADJUNCT, as (32) illustrates.

(32) rule(np(A), [np(B),pnct(C),np(D),pnct(E),conj(F),np(G)]).

$$\begin{bmatrix} \text{PRED} & \text{and} \\ \text{CONJ:1} & \begin{bmatrix} \text{PRED} & \text{....} \end{bmatrix} \\ \text{CONJ:2} & \begin{bmatrix} \text{PRED} & \text{....} \end{bmatrix} \\ \text{CONJ:3} & \begin{bmatrix} \text{PRED} & \text{....} \end{bmatrix} \end{bmatrix}$$

Ideally, then, the template must pick out the **conj** as the f-head and treat other categories, except for **pnct** as CONJ functions. However, consideration of the set of np rules involving coordination makes clear that things are unfortunately considerably more complicated. Because the treebank representations are extremely flat, the scope of coordination is not indicated by the presence of a distinct sub-tree: this means that it is not possible to assign all daughters (except **conj** and **pnct** to CONJ functions.

(33)
```
rule(np(A), [n0(B),conj(C),n0(D)]).
rule(np(A), [np(B),conj(C),np(D)]).
rule(np(A), [np(B),pnct(C),conj(D),np(E)]).
rule(np(A), [np(B),pnct(C),np(D),conj(E),np(F)]).
rule(np(A), [np(B),pnct(C),np(D),pnct(E),conj(F),np(G)]).
rule(np(A), [adj(B),conj(C),adj(D),n0(E),n0(F)]).
```

```
rule(np(A), [adj(B),conj(C),adj(D),n0(E),pp(F)]).
rule(np(A), [adv(B),num(C),adj(D),n0(E),n0(F),conj(G),n0(H)]).
rule(np(A), [det(B),adj(C),n0(D),conj(E),n0(F),n0(G)]).
rule(np(A), [det(B),adj(C),n0(D),conj(F),n0(G),pp(E)]).
rule(np(A), [det(B),adv(C),conj(D),adv(E),adj(F),n0(G),pnct(H),relcl(I)]).
rule(np(A), [det(B),n0(C),n0(D),pnct(E),n0(F),conj(G),n0(H)]).
rule(np(A), [posspron(B),n0(C),pp(D),conj(E),adv(F)]).
```

The `conj` macro is specialised for associating the right grammatical functions within the f-structure: however, in the statement of the template it is also necessary to ensure that the maximal applicable substring of daughters is covered. The `conj(List,M,CatTy,FTy,FD)` macro is defined for conjunctive patterns. Its first argument `List` is a list describing (a portion of) a rule RHS. `List` may contain regular expressions. `M` is a mother f-structure variable mentioned elsewhere in the `fd/3` template. `CatTy` is the category type of the conjuncts in the conjunctive structure. This is required to find the conjuncts whose f-structure variables contribute to the annotation to be constructed. `FTy` specifies the feature which is going to be used in the annotation. Finally, `FD` is the resulting feature description.

(34)
```
fd(rule(np(NP),Dtrs),
   [ element(conj(_),Dtrs),           %% <= just effi-
ciency .. i.e.
                                      %% don't search if there is no
     sequence(Dtrs,Left,Seq,Right),   %% conj
     not(element(n0(_),Left)),
     not(element(n0(_),Right)),
     eq(Seq,[conj(['kleene+'([n0(A),opt([pnct(_)])]),
                   conj(E),n0(A1)],NP,n0,_,FD)]) ],
    FD ).
```

Further complications are the coordination of adjectives directly under np and even of adverbs modifying adjectives under np. General statements can be written for these under which in each case the coordinate structure is identified and assigned the correct sort of ADJUNCT function in the mother f-structure.

(35)
```
fd(rule(np(NP),Dtrs),
   [ member(conj(_),Dtrs),             %% efficiency
     sequence(Dtrs,Left,Seq,Right),
     not(element(adj(_),Left)),
     not(element(adj(_),Right)),
     eq(Seq,[conj(['kleene*'([adj(A)]),adj(A2),conj(E),adj(A1)],
                 M,adj,_,FD)]) ],
   [ NP:np_adjunct:1 === M | FD ]).
```

4 Experiments

In this section, we report on experiments in automatically compiling the templates over the grammar rules and compare the results to our hand-coded grammar.

4.1 Experiment Design and Data

Our experiment involves the first 100 trees of the AP treebank (Leech and Garside 1991). We preprocess the treebank using the structure preserving grammar compaction method reported in (van Genabith et al. 1999b) and extract a treebank grammar following (Charniak 1996). The large number of highly discriminating terminal and non-terminal categories results in a large number of often very specific rules: the grammar compaction method provides a more general grammar that still preserves all important categorial information to drive automatic annotation. Compaction works by generalising tags, i.e. collapsing tags (and categories) into supertags. This reduces the number of rules from 509 to 330. The sentences in the fragment range from 4 to 50 tokens (including punctuation symbols). We develop a set of feature structure annotation templates. A template interpreter compiles the templates over the rules in the treebank grammar.

In order to evaluate the results of automatic annotation we manually constructed a reference grammar following (van Genabith et al., 1999a,b,c). The grammar features 1128 annotations, on average about 3.4 annotations per rule.[8]

4.2 Automatic Annotation and Evaluation

We constructed 129 templates, this against 330 CFG rules resulting in a template/rule ratio of 0.39. We expect the ratio to skew in favour of templates as we proceed to larger fragments. Automatic annotation generates 1108 annotations, on average about 3.7 annotations per rule. We evaluate the automatic annotation procedure in terms of *precision* and *recall*.

(36)

$$precision = \frac{\#generated\ annotations\ also\ in\ reference}{\#generated\ annotations}$$

$$recall = \frac{\#reference\ annotations\ also\ generated}{\#reference\ annotations}$$

	Experiment
precision	93.38%
recall	91.58%

These numbers, although good, are conservative: *precision* and *recall* are computed automatically and currently our annotation matcher is not complete.[9]

The results are encouraging and indicate that while automatic annotation is (slightly) more often partial than incorrect, a small number of annotation templates can be written for a grammar fragment which appears to be quite complex. The generalisations made are simple and robust, and can be expected to considerably ease the annotation burden on the grammar writer.

Having been confronted with 'real' text, we have been forced to distinguish a number of grammatical functions for which there is good c-structure evidence and/or motivation in real text, but which are not discussed

[8] Templates, grammars and f-structures are available at `http://www.compapp.dcu.ie/~away/Treebank/treebank.html`.

[9] `P1 = P2` $\models$ `P2 = P1` and `A=B, A:P1 = P2` $\models$ `B:P1 = P2` where `Pi` are paths and `A,B` variables; hence it misses `P1:P2 = P3, P1 = A` $\models$ `A:P2 = P3` type inferences.

in the theoretical literature. In particular, we have postulated a pre-modificational HEADMOD grammatical function within NPs. The biggest challenge presented by the very flat treebank representations concerns the coordination data, and in particular the interaction of coordination with other phenomena.

5 Conclusions and Further Work

We have presented and extensively exemplified a method for the automatic f-structure annotation of treebank grammars. At this stage our intent has been to present the methods and to explore some of their potential. The approach applies to a CFG, such as that derived from a treebank, and yields an annotated grammar, which can either be used to reparse treebank trees or serve as a basis for developing a stand-alone LFG resource. It uses a compaction technique for generalising overspecific categorisation. The structure of treebank entries remains unchanged. We implemented an order-independent annotation template interpreter. Order independence can ease development and maintainance of annotation principles, but requires more complex rule constraints.

Automatic annotation holds considerable potential in curtailing development costs and opens up the possibility of tackling large fragments. To date, our experiments are admittedly small-scale. Still, we have presented an important grammar development and treebank annotation methodology which is data-driven, semi-automatic and reuses existing resources. We found the LFG framework very conducive to our experiments. We do believe, however, that the methods can be generalised, and we intend to apply them in an HPSG scenario. Note further that our methods encourage work in the best linguistic tradition as (i) they are concerned with real language and (ii) they enforce generalisations in the form of annotation principles. The experiments show how theoretical work and ideas on principles can translate into grammar development for real texts. In this sense the methods bridge the often perceived gap between theoretically motivated views of grammar as a set of principles versus grammars for ‘real’ text.

Bibliography

Bod, R., and R. Kaplan. 1998. A Probabilistic Corpus-Driven Model for Lexical-Functional Analysis. In *Proceedings of COLING/ACL’98*, 145–151.

Bresnan, J. 2000. Lexical Functional Syntax. Forthcoming, Blackwells Publishers, Oxford.

Butt, M., T. H. King, M.-E. N. no, and F. Segond. 1999. *The Grammar Writer’s Cookbook.* Stanford: CSLI Publications.

Charniak, E. 1996. Tree-bank Grammars. In *AAAI-96. Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1031–1036. MIT Press.

Dalrymple, M. 1999. Lexical-Functional Grammar. Manuscript, Xerox PARC.

Frank, A., T. King, J. Kuhn, and J. Maxwell-III. 1998. Optimality Theory Style Constraint Ranking in Large-scale LFG Grammars. In M. Butt and T. King (Eds.), *Proceedings of the LFG98 Conference*, University of Queensland, Brisbane, CSLI Online Publications, Stanford, CA. `http://www-csli.stanford.edu/publications/`.

Frank, A. 2000. Automatic F-structure Annotation of Treebank Trees. In M. Butt and T. King (Eds.), *Proceedings of the LFG00 Conference*, CSLI Online Publications, Stanford, CA. `http://www-csli.stanford.edu/publications/`.

Kaplan, R., and J. Bresnan. 1982. Lexical Functional Grammar: a Formal System for Grammatical Representation. In J. Bresnan (Ed.), *The Mental Representation of Grammatical Relations*, 173–282. Cambridge, Mass: MIT Press.

King, T. H. 1995. *Configuring Topic and Focus in Russian.* Stanford: CSLI Publications.

Kroeger, P. 1995. *Phrase Structure and Grammatical Relations in Tagalog*. Stanford: CSLI.

Krotov, A., M. Hepple, R. Gaizauskas, and Y. Wilks. 1998. Compacting the Penn Treebank Grammar. In *Proceedings of COLING/ACL'98*, 699–703.

Leech, G., and R. Garside. 1991. *Running a Grammar Factory: On the Compilation of Parsed Corpora, or 'Treebanks'*. Mouton de Gruyter, Berlin. 15–32.

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

van Genabith, J., L. Sadler, and A. Way. 1999b. Structure Preserving CF-PSG Compaction, LFG and Treebanks. In *Proceedings ATALA Workshop - Treebanks*, Journées ATALA, Corpus annotés pour la syntaxe, Universite Paris 7, France, 18-19 Juin 1999, 107–114.

van Genabith, J., A. Way, and L. Sadler. 1999c. Semi-Automatic Generation of F-Structures from Tree Banks. In M. Butt and T. King (Eds.), *Proceedings of the LFG99 Conference*, Manchester University, 19-21 July, CSLI Online Publications, Stanford, CA. `http://www-csli.stanford.edu/publications/`.