

Data-Driven Compilation of LFG Semantic Forms

Josef van Genabith[◦], Louisa Sadler* and Andy Way[◦]

[◦]Dublin City University
Computer Applications
Dublin 9, Ireland

*University of Essex
Language and Linguistics
Colchester, U.K.

{josef, away}@compapp.dcu.ie louisa@essex.ac.uk

Abstract

In a recent paper (van Genabith et al., 1999) describe a semi-automatic method for annotating tree banks with high level Lexical Functional Grammar (LFG) f-structure representations. First, a CF-PSG is automatically induced from the tree bank using the method described in (Charniak, 1996). The CF-PSG is then manually annotated with functional schemata. The resulting LFG is then used to deterministically “reparse” the original tree bank representations simply following the c-structure defined by the original annotations, thereby inducing f-structures corresponding to the original c-structures. The annotated grammars, however, are not yet stand-alone LFGs. They cannot be used to analyse freely occurring strings as opposed to strings annotated with c-structures as in the original tree bank. What is missing is the LFG account of subcategorization in terms of *semantic forms* and *completeness* and *coherence* constraints. In the present paper we develop automatic and semi-automatic methods for compiling LFG semantic forms from the tree banks annotated with f-structure representations, effectively turning the grammars developed in the original approach into stand-alone LFG grammars. In addition, the method provides full semantic forms as PRED values for the f-structures obtained from the input tree bank.

1 Introduction

High quality training corpora are crucial for statistical approaches to natural language processing. For probabilistic unification grammars e.g. (Bod and Kaplan, 1998), corpora with feature structure annotations are required. Such corpora tend to be hard to come by and expensive to construct. The standard approach to constructing such re-

sources (apart from hand coding) is to develop a large-scale unification grammar and to parse input text. Given a wide coverage grammar, for each sentence in the input text the parser is likely to come up with hundreds or thousands of candidate analyses. From these a human expert has to pick the analysis with the closest fit for the given input string for inclusion in the corpus. This is time-consuming, error-prone and requires extensive linguistic expertise. In a recent paper (van Genabith et al., 1999), we propose a semi-automatic approach which avoids manual inspection of alternative parses. As input the method requires a tree bank. From this a CF-PSG is extracted using the method of (Charniak, 1996). The CF-PSG is manually annotated with Lexical Functional Grammar (Kaplan and Bresnan, 1982) functional schemata. The resulting LFG is then used to “reparse” the original tree bank representations simply following the c-structure defined by the original annotations, thereby inducing f-structures corresponding to the original c-structures. In this approach, the only possible source of non-determinism are the functional schemata and these are deterministic in the grammars in (van Genabith et al., 1999).

It turns out, however, that the grammars developed in this approach are not yet stand-alone LFGs. What is missing is the LFG account of subcategorization in terms of *semantic forms* and *completeness* and *coherence* constraints. This means that the grammars cannot be used for parsing strings (as opposed to annotated tree bank entries). In the present paper we describe automatic and semi-automatic methods to compile semantic forms from the f-structure annotated tree banks generated by (van Genabith et al., 1999), thereby turning the grammars developed in that approach into stand-alone Lexical Functional Grammars. The basic idea is simple: for each f-structure generated in the original approach, for each level of embedding we determine the local PRED value and collect the subcategorizable grammatical functions present at that level of embedding. From these we construct *proper* semantic forms.

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

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

Figure 1: Tree Bank Entry, Prolog Term Representation and Extracted Grammar

The method depends, of course, on the quality of the input f-structures. A syntactic dimension which is particularly sensitive in this respect is whether certain constituents are analyzed as adjuncts or as subcategorized grammatical functions. In the original grammars developed in (van Genabith et al., 1999), for each rule to which the distinction applies, the decision was made by inspecting occurrences of the rule in question in the tree bank. In a first experiment we ran our semantic form compiler on the output of this grammar. In a second experiment we extended the original annotations of those rules that, given the input tree bank, produce at least one adjunct / subcategorizable grammatical function misclassification with an alternative, one for the adjunct, one for the subcategorizable grammatical function. This move introduces a limited amount of non-determinism in parsing the tree bank. For each tree bank entry we manually select the best f-structure from the alternatives produced by the new grammar for inclusion in a corpus. From this corpus we then compile semantic forms.

The paper proceeds as follows: first, we describe the method detailed in (van Genabith et al., 1999); second, we briefly recall the treatment of subcategorization in LFG; third, we describe the semantic form compiler and report on our first experiment; fourth, we report on the second experiment; fifth, we evaluate the results obtained so far; sixth, we outline work in progress and finally, we conclude.

2 Semi-Automatic Generation of F-Structures from Tree Banks

The original experiment described in (van Genabith et al., 1999) is based on the publicly available fragment (the first 100 sentences) of the AP tree bank developed at the University of Lancaster (Garside and McEnery, 1993). The corpus is tagged, post-edited and hand parsed. The number of preterminal categories (tags) in the AP corpus is 183. The total number of non-lexical categories is 53. The longest sentence in the fragment contains 43 words, the shortest 4, with an average of about 20. The corpus is “skeletonally parsed”, where some partial parsing (unlabelled brackets) is permitted. Such unlabelled brackets indicate that some sequence forms some kind of constituent. In a pre-editing step such brackets

are removed in the approach in (van Genabith et al., 1999).

The tree bank entries are fed into a pre-compiler that transforms the corpus data structures into corresponding Prolog terms. The Prolog terms are then processed by a CF-PSG grammar extraction module which reads off the CF-PSG rules by recursively traversing local sub-trees in the corpus entries, following the technique documented in (Charniak, 1996). The initial processing steps are illustrated with a simple example from the corpus (first, the original corpus entry; second, the Prolog term; third, the output of the CF-PSG extraction module) in Figure 1 above.

LFGs (Kaplan and Bresnan, 1982) are constraint grammars with a CF-PSG backbone. The constraints are referred to as functional schemata. They are statements in an equality logic describing f-structures and, given a parse tree, define a correspondence between f-structure and CF-PSG nodes. In (van Genabith et al., 1999) we model this by using a simple version of graph unification described in (Gazdar and Mellish, 1989).¹ Lexical categories in the AP tag set are associated with macros. Those corresponding to the lexical entries in (1) are:

- (1)
- ```
macro(at(Word),FStr) :-
 FStr:spec === Word.

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

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

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

where each macro defines f-structure fragments for the set of words in the tag class. The rule annotations are direct translations of the corresponding

<sup>1</sup>This version provides a subset of the constraint language of LFG. Essentially it supports positive constraints over finite paths. It does not provide negation, regular path expressions or constraining ‘=<sub>c</sub>’ equations.

```
(2) rule(n(A), [at(B), nm1(C)]) :-
 A === B,
 A === C.

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

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

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

A “rearsing” interpreter then reparses the annotated treebank entries by following the tree annotations provided by the original annotators. In so doing the interpreter solves the constraint equations associated with the grammar rules and lexical macros involved in the parse, returning a single f-structure. For the example sentence in Figure 1 (previous page) the output of the “rearsing” interpreter yields:

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

From the first 100 sentences of the AP tree bank we compile 516 CF-PSG rules and 825 lexical entries. The complete set of f-structures obtained is documented in (Way et al., 1999).

### 3 Semantic Forms and Subcategorization

In LFG, subcategorization requirements are expressed in semantic forms (which list the subcategorized arguments of a predicate) and enforced by means of well-formedness constraints which hold at f-structure over subcategorizable grammatical functions. As an example, consider the following sentence from the AP corpus and the f-structure induced by the method of (van Genabith et al.,

(4)

```
tree(a001, '43', v, sent(n(nn2(police)),
v(vvd(checked), n(at(the), nm1(coliseum)),
p(if(for), n(nn2(bombs))), p(ics(before),
n(at(the), nm1(march)))))).

subj : pred : police
 num : pl
obj : spec : the
 pred : coliseum
 num : sg
obl : obj : pred : bombs
 num : pl
 pred : for
vp_adjunct : obj : spec : the
 : pred : march
 : num : sg
 : pred : before

tense : past
pred : checked
```

In LFG the semantic form associated with the type of **checked** at stake is CHECK( $\uparrow$  SUBJ,  $\uparrow$  OBJ,  $\uparrow$  OBL<sub>FOR</sub>) which states that **checked** requires a SUBJ, an OBJ and an OBL<sub>FOR</sub> argument. Subcategorization requirements (such as these) are enforced in terms of completeness and coherence constraints:

An f-structure is *locally complete* iff it features all grammatical functions specified in the semantic form of its local PRED feature. An f-structure is *complete* iff it is locally complete and all its subsidiary f-structures are locally complete. An f-structure is *locally coherent* iff the only subcategorizable grammatical functions featured at that level are those listed in the semantic form value of the local PRED feature. An f-structure is *coherent* iff it is locally coherent and all its subsidiary f-structures are locally coherent.

Subcategorizable grammatical functions are those which can appear in semantic forms while non-subcategorizable cannot. Notice that the PRED values in the example above (and indeed in all output of the the original approach reported in (van Genabith et al., 1999)) do not give any subcategorization information. This means that the LFG grammar obtained can only be used to “reparse” syntactically annotated strings in the original tree bank and associate these with f-structures. However, the resulting grammars cannot be used to parse unannotated strings: subcategorization requirements are not checked.

### 4 Compiling Semantic Forms: Experiment 1

In order to obtain a stand-alone LFG we need full semantic forms as PRED values together with

<sup>2</sup>In the grammars developed in (van Genabith et al., 1999) we use several non-standard function names e.g. `vp_adjunct`, `np_adjunct`, `headmod`. These retain potentially useful c-structure information.

an implementation of completeness and coherence checks. Given the f-structure annotations produced in (van Genabith et al., 1999), semantic forms can be compiled automatically as follows:

Given a set of subcategorizable grammatical functions, for each f-structure and for each level of embedding in those f-structures, determine the **pred** value at that level and collect the subcategorizable grammatical functions present at that level.

In other words, we can simply read off proper semantic forms from our f-structures, just as we are able to read off a CF-PSG from the original tree bank, using the method of (Charniak, 1996). Once we have proper semantic forms we automatically associate them with the corresponding lexical entries and together with our implementation of completeness and coherence we enforce LFG subcategorization requirements in parsing strings. This means that we have obtained a stand-alone LFG.

Given the example f-structure in (4), the Prolog implementation of our semantic form compiler yields:<sup>3</sup>

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

Of course, the method can only be applied with any degree of confidence if the input f-structures produced in (van Genabith et al., 1999) are correct. A syntactic dimension which is particularly sensitive in this respect is the distinction between subcategorizable grammatical functions and adjuncts, more precisely whether a given constituent is analyzed as a subcategorizable grammatical function or as an adjunct. In the original grammar in (van Genabith et al., 1999), for each rule where the distinction applies a decision was made as to whether (say) a prepositional phrase is analyzed as an adjunct or as a subcategorizable grammatical function based on analysis of occurrences of the rule in question in the original corpus. This approach yields surprisingly good results but is not a general solution.<sup>4</sup> Below, we list example semantic forms compiled from the f-structures associated with the tree bank fragment. The fragment gives rise to about 700 semantic forms. The high number of semantic form types is due to the fact that the predicates are derived from surface form. Morphological analysis to roots would significantly reduce the number.

<sup>3</sup>The semantic form compiler sorts arguments lexically. “:” is the main functor of `obl:for`, hence such arguments appear last.

<sup>4</sup>One reason why the results are good is that the rules extracted from the original tree bank tend to be extremely specific and apply to very few instances only.

The examples below list some verbs involving `obl` arguments and their corresponding prepositions:

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

What is of importance here is, of course, not the examples themselves but the way they were determined from the input corpus fragment. The following set contains nominals involving `obl` arguments:

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

## 5 Experiment 2

We have implemented a semantic form compiler and run an experiment on the output of the original deterministic system in (van Genabith et al., 1999). Of course, the f-structures produced in (van Genabith et al., 1999) do not always get it right. In some cases, they analyze what should be an adjunct as a subcategorized argument and vice versa. Such mistakes, are of course, reflected in the semantic forms compiled from the input f-structures. It turns out, however, that such cases are surprisingly rare: in the original deterministic approach roughly 5% of the semantic forms compiled turned out to be incorrect.<sup>5</sup> At the present stage we can only speculate as to why this is the case. Rule specificity is a determining factor. The CF-PSG rules compiled from the tree bank are extremely specific and apply to only very few input strings. Often simple inspection of instances where a given rule applies in the input tree bank fragment is sufficient to determine whether (say) a prepositional phrase in the rule is best annotated as providing an adjunct or a subcategorizable grammatical function. To a certain extent this is an artefact of the small size of the tree bank fragment available for the experiment. There is certainly no obvious asymmetry between subcategorizable grammatical function analyses and

<sup>5</sup>Note that some of the semantic forms classified as correct will not contain subject functions. This is discussed further in section 6 below.

non-subcategorizable grammatical function analyses: a single correct subcategorizable grammatical function analysis in the input corpus is sufficient for the semantic form compiler to establish the correct semantic form, even if all other analyses for the same predicate are incorrect. On the other hand, a single incorrect analysis of what really is a non-subcategorizable grammatical function as a subcategorizable grammatical function will yield an incorrect semantic form. To give a simple example, consider sentence 22 from the corpus in Figure 3 (in the Appendix) where the first f-structure (generated by the original rule set in experiment 1) incorrectly analyses the prepositional phrase **from a&t university in greensboro** as a subcategorized grammatical function.

In order to improve the quality of our input f-structures in a second experiment we deliberately introduce a limited amount of non-determinism in the original grammar in (van Genabith et al., 1999). For each rule which at least once incorrectly classifies a grammatical function as an adjunct or vice versa we introduce a local disjunction between a subcategorizable grammatical function and an adjunct analysis and “reparse” the original tree bank fragment. Consider the initial noun phrase **n** and the corresponding rule involved in Figure 3 in the Appendix. The first instance of the rule below gives the original version where the prepositional phrase **p** is analyzed as an **obl**. The second instance of the rule introduces a local disjunction between an **obl** and an **np\_adjunct** analysis of the prepositional phrase:

```
(8) rule(n(A), [nn2(B), p(C)]) :-
 A === B,
 A:obl === C.

rule(n(A), [nn2(B), p(C)]) :-
 A === B,
 (A:obl === C ;
 A:np_adjunct === C).
```

Note that if  $n$  disjunctive rules are involved in “reparsing” a tree bank representation and if each such rule encodes a binary choice, the parse will yield worst case  $2^n$  results.<sup>6</sup> Both the first and the second f-structure in our example in Figure 3 are generated by the disjunctive rule in (8). From the results of the disjunctive rule set we manually select the best fit given the input tree bank representation for inclusion in the c- and f-structure corpus. From this corpus we then compile semantic forms. In this approach, about 1% of the semantic forms compiled are incorrect.

## 6 Results Obtained

In both experiments the results obtained depend entirely on the quality of the input f-structures.

<sup>6</sup>The worst case proviso is there since some parses will be mutually exclusive due to feature clashes.

The difference between experiment 1 and 2 is that in experiment 2 we tried to improve the quality of the input f-structures by deliberately introducing a limited amount of disjunction into our functional annotations. As a consequence, in this experiment we have to hand pick the best analysis for inclusion in the f-structure annotated tree bank from which semantic forms are read off. In our experiment, the disjunction introduced was very limited and did not lead to a combinatorial explosion. This is entirely due to the fact that the CF-PSG compiled from the original tree bank is extremely specific with rules applying to very few cases only. It is expected that this is going to change somewhat, though not drastically, if we move to a larger fragment. Even though in such a scenario the  $\# \text{ rule} / \# \text{ local tree}$  ratio is going to decrease (resulting in a reduced rule specificity), results obtained by (Charniak, 1996) and (Krotov et al., 1998) indicate that the number of CF-PSG rules obtained from similar hand-coded tree banks grows steadily with the number of tree bank entries.

The quality of the f-structures generated in our approach depends on the constituent structure tree analyses provided by the original annotators and on the f-descriptions we ourselves provide. If the original annotators misparsed a sentence, it is hard to correct this using functional annotations. The only long-term approach would be to relabel the input trees, an option we have so far not pursued. As an example consider the tree bank entry for sentence 36 in Figure 2 in the Appendix. Here the first prepositional phrase **p** is incorrectly analyzed as a daughter of the verb phrase **v** rather than as a constituent of the noun **n(nn(membership))**. Disjunctive f-structure annotations will incorrectly classify the prepositional phrase as a subcategorizable grammatical function or as adjunct to the verb (rather than the noun) and this is exactly what happens with our rule annotations.

The f-structure annotations we provide in (van Genabith et al., 1999) are not perfect. The f-structures listed in Figure 3 in the Appendix are cases in point. Both f-structures do not reflect the fact that the subject of the matrix control verb **try** is supposed to be token identical with the subject of the verb in the subordinate clause (the **xcomp**). A simple control equation in the lexical entry of **try**

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

would be sufficient. However, in our current approach lexical f-structures are induced automatically on the basis of macros indexed by tag class (**vvd** in the case at hand) and the tags provided by the AP tree bank do not distinguish control verbs. In order to provide the required control equa-

tions we would have to manually associate certain lexical entries with the required constraints. In the case at hand (Figure 3) this means that the semantic form compiler will extract a semantic form `drown([obj,obl:with])` for the phrasal verb `drown out` in the embedded clause which does not feature a `subj` function. A similar situation applies with coordinate constructions. The current version of our grammar does not distribute e.g. subjects into verb phrase coordinate structures. For string parsing (rather than tree bank entry “rearsing”) with global completeness and coherence constraints based on the semantic forms we currently extract this means that we have to ignore subject requirements in semantic forms.

## 7 Work in Progress

We are currently experimenting with grammar compaction techniques to improve the quality of the CF-PSG base we extract from the original tree bank following the method of (Charniak, 1996). It is helpful to distinguish between two types of grammar compaction techniques: the first is *structure preserving* while the second is not. The method presented in (Krotov et al., 1998) is an instance of a *structure changing* technique. Essentially, rules whose right-hand-sides can be parsed by other rules are replaced. As an alternative, we have developed a structure-preserving grammar compaction technique. The basic idea is simple: the AP tag set (the set of lexical categories) is quite fine-grained. In (van Genabith et al., 1999) we developed a set of macros indexed by tag categories which encode f-structure information relevant to the tag class. In doing so we have shifted information originally encoded in the tag category (i.e. in a CF-PSG lexical category) into the feature-structure encoding. This means that we can now collapse the original set of AP tree bank tag categories into a much smaller set of super-tags. We first automatically distribute the lexical macros over our lexical entries and then automatically relabel the lexical entries thus obtained and the original tree bank with the super-tags. From the relabelled tree bank we induce a reduced CF-PSG which now has fewer but more general rules. We annotate the CF-PSG essentially by merging the functional annotations of the original CF-PSG. The grammars are used to “reparse” the relabelled tree bank in order to induce f-structures and to parse free strings. Initial experiments have shown that the resulting grammars are highly non-deterministic even if used to “reparse” the relabelled tree bank. We hope to control some of the non-determinism by enforcing completeness and coherence constraints using the semantic forms obtained in the approach detailed in the present paper.

## 8 Conclusion and Further Work

In the present paper we have extended the method described by (van Genabith et al., 1999) with automatic and semi-automatic (experiment 2) compilation of LFG semantic forms yielding stand-alone LFG grammars obtained from tree banks. Given tree banks, (Charniak, 1996) was able to compile CF-PSGs. Given c- and f-structure annotations in (van Genabith et al., 1999), we were able to compile LFG semantic forms. Viewed from a different angle, the paper is part of a general effort to map tree banks into richer linguistic representations.

## Acknowledgments

We would like to thank our anonymous referees for helpful and inspiring comments and feedback. Remaining mistakes are our own.

## References

- R. Bod and R. Kaplan. A Probabilistic Corpus-Driven Model for Lexical-Functional Analysis. In COLING-ACL’98: Montreal, Canada, 1:145–151, 1998.
- E. Charniak. Tree-bank grammars. In AAAI-96, Proceedings of the Thirteenth National Conference on Artificial Intelligence, MIT Press, pp.1031-1036, 1996.
- R. Garside and A. McEnery. Treebanking: the Compilation of a Corpus of Skeleton-Parsed Sentences. In: E. Black, R. Garside & G. Leech (eds) *Statistically-driven computer grammars of English: The IBM/Lancaster approach*, Rodopi, Amsterdam, 1993.
- G. Gazdar and C. Mellish. *Natural Language Processing in Prolog: an Introduction to Computational Linguistics*. Addison-Wesley, Wokingham, UK, 1989.
- R.M. Kaplan and J. Bresnan. Lexical Functional Grammar. In Bresnan, J., editor 1982, *The mental representation of grammatical relations*. MIT Press, Cambridge Mass. 173–281, 1982.
- A. Krotov, M. Hepple, R. Gaizauskas and Y. Wilks. Compacting the Penn Treebank Grammar. In COLING-ACL’98, Montreal, Canada, pp.699-703, 1998.
- J. van Genabith, A. Way and L. Sadler. Semi-Automatic Generation of F-Structures from Tree Banks. LFG99, The fourth International Conference on LFG. Manchester, U.K., 19-21 July 1999.
- A. Way, L. Sadler and J. van Genabith. Mapping the AP Treebank to f-Structures: an Experiment and Initial Results. Working Paper, Dublin City University and the University of Essex, 1999.

## Appendix

```
sent(n(mc(fourteen),nn2(persons),pnct(',') ,tg(n(dat(most)),v(vvg(claiming),n(nn(membership))),
p(ii(in),n('n&'(at(the),np1(ku),np1(klux),nnj(klan)),cc(or),`n+'(at(the),jj(nazi),nnj(party)))))),
pnct(',') ,v(vv0(face),n(nn2(charges),p(ii31(in),ii32(connection),ii33(with),
n(at(the),nn2(shootings))))))
```

```
subj : relmod : subj : spec : most
 obj : pred : membership
 obl : obj : conj : 1 : headmod : headmod : pred : ku
 num : sg
 pred : klux
 num : sg
 spec : the
 pred : klan
 2 : headmod : pred : nazi
 spec : the
 pred : party
 pred : or
 pred : in
 pred : claiming
 spec : fourteen
 num : pl
 pred : persons
obj : obl : obj : obl : obj : spec : the
 pred : shootings
 num : pl
 pred : with
 pred : connection
 pred : in
 pred : charges
 num : pl
pred : face

subj : relmod : subj : spec : most
 obj : pred : membership
 vp_adjunct : obj : conj : 1 : headmod : headmod : pred : ku
 num : sg
 pred : klux
 num : sg
 spec : the
 pred : klan
 2 : headmod : pred : nazi
 spec : the
 pred : party
 pred : or
 pred : in
 pred : claiming
 spec : fourteen
 num : pl
 pred : persons
obj : np_adjunct : obj : obl : obj : spec : the
 pred : shootings
 num : pl
 pred : with
 pred : connection
 pred : in
 pred : charges
 num : pl
pred : face
```

Figure 2: Sentence 36

```
sent('s&'(n(nn2(students),p(ii(from),n(np1('a&t'),nn1(university),p(ii(in),n(np1(greensboro)))))),
v(vvd(tried),ti(to(to),vv0(drown),rp(out),n(at(the),nn2(speakers)),p(iw(with),n(nn2(chants)))))),
pnct(', '),cc(and), 's+'(n(at(the),nn1(march)),v(vvd(resumed))))
```

```
conj : 1 : subj : obl : obj : headmod : pred : a&t
 num : sg
 np_adjunct : obj : pred : greensboro
 num : sg
 pred : in
 pred : university
 num : sg
 pred : from
 pred : students
 num : pl
xcomp : part : pred : out
 obj : spec : the
 pred : speakers
 num : pl
 obl : obj : pred : chants
 num : pl
 pred : with
 to : +
 inf : +
 pred : drown
tense : past
pred : tried
2 : subj : spec : the
 pred : march
 num : sg
tense : past
pred : resumed
pred : and

conj : 1 : subj : np_adjunct : obj : headmod : pred : a&t
 num : sg
 np_adjunct : obj : pred : greensboro
 num : sg
 pred : in
 pred : university
 num : sg
 pred : from
 pred : students
 num : pl
xcomp : part : pred : out
 obj : spec : the
 pred : speakers
 num : pl
 obl : obj : pred : chants
 num : pl
 pred : with
 to : +
 inf : +
 pred : drown
tense : past
pred : tried
2 : subj : spec : the
 pred : march
 num : sg
tense : past
pred : resumed
```

Figure 3: Sentence 22