Ambiguity Management in Grammar Writing

Tracy Holloway King[1], Stefanie Dipper[2], Anette Frank[3],
Jonas Kuhn[2], John Maxwell[1]

Abstract

When lingusitically motivated grammars are implemented on a larger scale, and applied to real-life corpora, keeping track of ambiguity sources becomes a difficult task. Yet it is of great importance, since unintended ambiguities arising from underrestricted rules or interactions have to be distinguished from linguistically warranted ambiguities. In this paper we report on various tools in the XLE grammar development platform which can be used for ambiguity management in grammar writing. In particular, we look at packed representations of ambiguities that allow the grammar writer to view sorted descriptions of ambiguity sources. Also discussed are tools for specifying desired tree structures and for cutting down the solution space prior to parsing.

1 Introduction

Xerox PARC has developed the XLE system (Xerox Linguistic Environment), a platform for large-scale LFG (Lexical-Functional Grammar) grammar development. XLE comprises interfaces to finite-state preprocessing modules for tokenization and morphological analysis, as well as an efficient parser and generator for LFG grammars. Since 1995, the PARGRAM (Parallel LFG Grammar Development) project, a joint initiative of Xerox PARC, XRCE Grenoble, and the University of Stuttgart (IMS), has investigated the potential of LFG for large-scale NLP applications. PARGRAM encompasses linguistic research in LFG-based parallel grammar development for different languages: English, French, and German; recently, the University of Bergen as a new partner has started developing a Norwegian grammar, and a Japanese grammar is being developed by Fuji Xerox.

LFG (Bresnan (1982, 2000)) is particularly well suited for high-level syntactic analysis in multilingual NLP tasks. The LFG formalism assigns natural language sentences two levels of linguistic representation — a constituent phrase structure (c-structure) and a functional structure (f-structure) — which are related in terms of a functional projection, or correspondence function (ϕ–projection). The c-structure encodes constituency (dominance) and surface order (precedence). The f-structure is an attribute-value representation which encodes syntactic information in terms of morphosyntactic features (NUM, GEND, TENSE, etc.) as well as functional relations between predicates and their arguments or adjuncts. The two levels of representation are related via the correspondence function ϕ, which maps partial c-structures to partial f-structures. Documentation and discussion of the implemented English, French, and German LFG grammars are given in Butt et al. (1999).

In this paper we focus on a specific, complex task in practical, large-scale LFG grammar development — the management of ambiguities — and how it is supported via various tools and underlying processing techniques in the XLE system. In doing so, we also discuss how the LFG/XLE specific tools are more generally applicable to the grammar writing task, regardless of theoretical framework or platform.

[1]NLTT/ISTL, Xerox PARC, 3333 Coyote Hill Rd, Palo Alto, CA 94304 USA
{thking,maxwell}@parc.xerox.com
[2]Institut für maschinelle Sprachverarbeitung, Universität Stuttgart, Azenbergstraße 12, 70174 Stuttgart, GERMANY
{dipper,kuhn}@ims.uni-stuttgart.de
[3]Xerox Research Centre Europe, 6, Chemin de Maupertuis, 38240 Meylan, FRANCE
anette.frank@xrce.xerox.com

The paper is organized as follows: Section 2 presents the ambiguity problem in large-scale grammar development and how the processing complexity of ambiguity is approached in the XLE system, focussing in particular on the notion of ambiguity packing. Section 3 presents some typical grammar writing tasks that involve ambiguity management and how they are supported in the XLE system. We introduce the main interfaces of the XLE system and first present some basic facilities for viewing and selecting analyses based on c-structure descriptions in sections 4.1 and 4.2; in section 4.3 we describe more advanced facilities and views, which allow the grammar writer to select structures from a single, packed f-structure. Sections 5 and 6 present additional devices for detecting ambiguity sources in a grammar and for reducing the search space of possible analyses for grammar/analysis inspection. Finally, section 7 provides a summary with some discussion.

2 Ambiguity management in grammar development

2.1 Ambiguity

Ambiguity is one of the main problems faced by large-scale computational grammars. Ambiguities can arise through rule interactions, via alternative definitions of lexical entries, or simply from linguistically justified syntactic ambiguities. As opposed to human interpreters, computational grammars are not yet able to correctly determine the contextually correct or intended syntactic analysis from a set of alternative analyses. Thus, a computational grammar which covers a realistic fragment of natural language will, for a given sentence, come up with a large number of possible analyses, most of which are not perceived by humans or are considered inappropriate in the given context.

There are several aspects of managing ambiguity in language processing (see Carter (1997) and references therein on tools for ambiguity management). (i) The parsing and generation algorithms have to deal with an exponential number of ambiguous structures in an efficient way, avoiding combinatorial explosion; and (ii) some disambiguation strategy has to adopted, depending on the application of the grammar (a particular strategy may be to preserve ambiguity wherever possible, e.g., in machine translation; in other application contexts, one might want to apply disambiguation strategies). These two aspects have received a lot of attention in the literature. Here, we want to address a third aspect that is independent of these two: while developing a linguistically motivated grammar, it is important to keep track of the ambiguity sources. Unlike more shallow approaches which typically conflate the tasks of parsing and disambiguation, a deep grammar assigns all the syntactically available readings to a given string. When dealing with real-life sentences, sophisticated methods of managing these ambiguities are thus required.

2.2 Packing ambiguities

The parsing and generation algorithms realized in XLE are based on insights from research into efficient processing algorithms for unification-based grammars (see in particular Maxwell and Kaplan (1989, 1993, 1996) and Shemtov (1997)).[4] One important facet of these efficient processing techniques is an algorithm for contexted constraint satisfaction, a method for processing ambiguities efficiently in a chart-like "packed" representation.

A major source of computational complexity with higher-level syntactic grammars is the high potential

[4]The unification algorithm described in Maxwell and Kaplan (1996) takes advantage of simple context-free equivalence in the feature space. As a result, sentences parse in cubic time in the typical case, though still being exponential in the worst case.

for ambiguities, especially with large-coverage grammars. While disjunctive statements of linguistic constraints allow for a transparent and modular specification of linguistic generalizations, the resolution of disjunctive feature constraint systems is expensive, in the worst case exponential. Conjunctive constraint systems, on the other hand, can be solved by standard unification algorithms which do not present a computational problem.

In standard approaches to disjunctive constraint satisfaction, disjunctive formulas are therefore converted to disjunctive normal form (DNF), as in (1). Conjunctive constraint solving is then applied to each of the resulting conjunctive subformulas. However, the possibly exponential number of such subformulas results in an overall worst-case exponential process. Moreover, in conversion to DNF individual facts are replicated in several distinct conjunctive subformulas. This means that they have to be recomputed many times.

(1)
$$(a \lor b) \land x \land (c \lor d) \quad \overset{\text{DNF}}{\Rightarrow} \quad \begin{array}{l} (a \land x \land c) \\ \lor\ (a \land x \land d) \\ \lor\ (b \land x \land c) \\ \lor\ (b \land x \land d) \end{array}$$

Maxwell and Kaplan (1989) observe that, although the number of disjunctions to process grows in rough proportion to the number of words in a sentence, most disjunctions are local and independent of each other. The general pattern is that disjunctions that arise from distinct parts of the sentence do not interact, as they are embedded within distinct parts of the f-structure. If disjunctions are independent, they conclude, it is not necessary to explore all combinations of disjuncts as they are rendered in DNF, in order to determine the satisfiability of the entire constraint system.

On the basis of these observations, Maxwell and Kaplan (1989) devise an algorithm for contexted constraint satisfaction — realized in the XLE parsing and generation algorithms — that reduces the problem of disjunctive constraint solving to the computationally much cheaper problem of conjunctive contexted constraint solving. The disjunctive constraint system is converted to a contexted conjunctive form (CF), a flat conjunction of implicational (contexted) facts, where each fact (a, b, x, . . .) is labeled with a propositional (context) variable p, q or its negation, as in (2)

(2)
$$(a \lor b) \land x \land (c \lor d) \quad \overset{\text{CF}}{\Rightarrow} \quad (p \rightarrow a) \land (\neg\, p \rightarrow b) \land x \land (q \rightarrow c) \land (\neg\, q \rightarrow d)$$

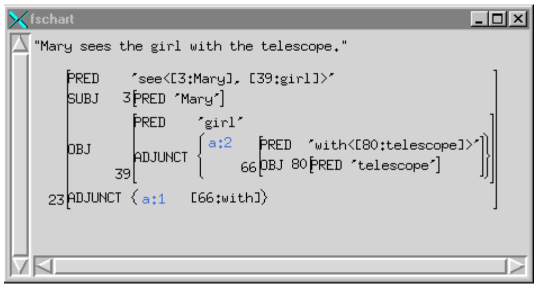
based on the Lemma in (3):

(3) $\phi_1 \lor \phi_2$ is satisfiable
iff $(p \rightarrow \phi_1) \land (\neg p \rightarrow \phi_2)$ is satisfiable, where p is a new propositional variable.

Context variables p and their negations are thus used to specifiy the requirement that for a disjunction of facts $\phi_1 \lor \phi_2$ at least one of the disjuncts is true.

As can be seen in the above example, conversion to CF has the advantage that each fact appears only once, and thus will be processed only once. The resulting formula is a flat conjunction of implicational facts, which forms a boolean constraint system that can be solved efficiently, based on mathematically well-understood, general and simple principles (see Maxwell and Kaplan (1989) for details).

What is important for our concerns is that disjunctive constraint processing allows for the representation of a set of ambiguous f-structures in a single, packed f-structure, a so-called f-structure chart, where disjunctive facts are not compiled out and duplicated. In the packed f-structure (chart) attribute-values are indexed with their corresponding context variables, as displayed in Fig. 1.[5]



```
"Mary sees the girl with the telescope."

    PRED     'see<[3:Mary], [39:girl]>'
    SUBJ   3 PRED 'Mary'
               PRED     'girl'
                          a:2    PRED  'with<[80:telescope]>'
    OBJ        ADJUNCT {      66 OBJ 80 PRED 'telescope'
            39
  23ADJUNCT { a:1   [66:with]}
```

Figure 1: F-structure chart for *Mary sees the girl with the telescope.*

In the given example, the ambiguity resides in the attachment level of the PP as a VP- or NP-adjunct, as seen graphically in the trees in Fig. 2.

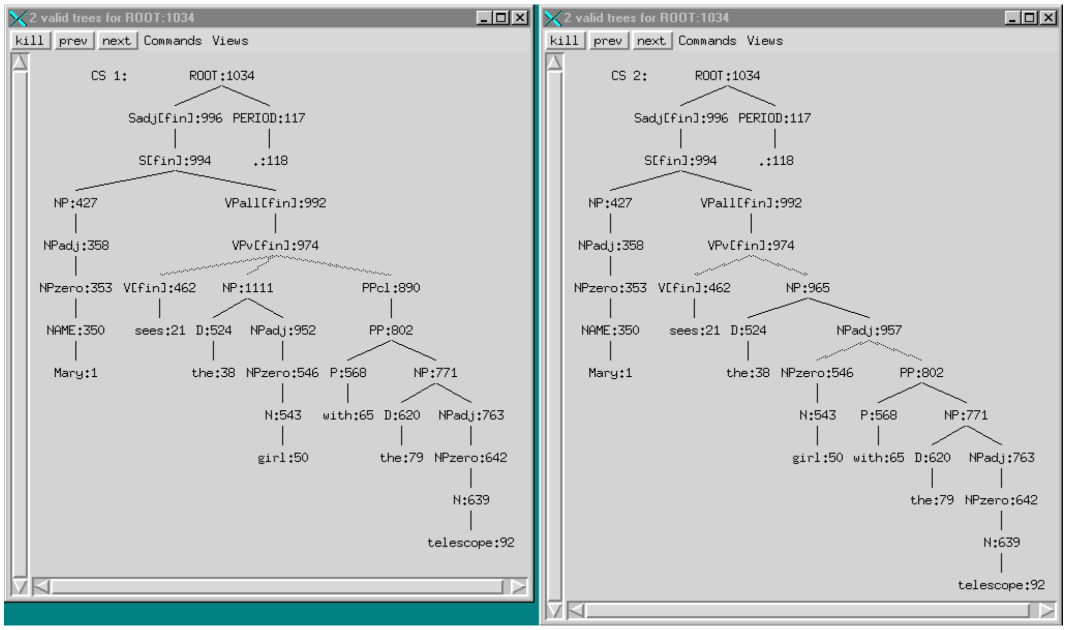


Figure 2: Trees for *Mary sees the girl with the telescope.*

While this ambiguity affects the entire c-to-f-structure mapping down from the level of VP, it is captured by the single local disjunctive contexts a_1 and a_2 in the f-structure chart displayed in Fig. 1. Context a_1 specifies the PP identified by f-structure node f_{66} as an element of the ADJUNCT set in the main predicate's f-structure f_{23}. In context a_2, the PP is specified as an element of the OBJect's (f_{39}) ADJUNCT set. All remaining f-structure constraints are conjoined in the TRUE context.

(4)
$$\begin{array}{lll} a_1 & \rightarrow & f_{66} \in (f_{23}\ \text{ADJUNCT}) \\ a_2 & \rightarrow & f_{66} \in (f_{39}\ \text{ADJUNCT}) \\ \text{TRUE} & \rightarrow & (f_{23}\ \text{OBJ}) = f_{39} \\ \text{TRUE} & \rightarrow & (f_{23}\ \text{PRED}) = \text{`see}\langle(\uparrow\text{SUBJ})\ (\uparrow\text{OBJ})\rangle\text{'} \ldots \end{array}$$

The local disjuncts are directly accessible through their context variables. That is, we can select one or

[5]The f-structure in Fig. 1 shows only the PRED values and no other attributes. For a more complete f-structure, see Fig. 5.

the other reading from the chart by selecting (clicking) its corresponding context variable, implicitly setting it to TRUE.

3 Grammar Writing Tasks

For the purposes of this paper we focus on two major tasks faced by grammar writers when there are multiple outputs of the parser. The first task is to check whether a particular desired structure is contained within the output. The second is to determine which structures in the output are undesirable overgeneration. Consider a sentence like (5).

(5) I want him to leave.

With a relatively large-scale grammar, this sentence might have three analyses: one in which *want* takes three arguments (subject, object, and infinitive) and two in which it takes two arguments (subject and object) and an adjunct (infinitive or prepositional phrase, e.g., *I want the box on the shelf*).

If, for example, the grammar writer has just added three-argument verbs of this type, it is necessary to search through the set of output structures to make sure that the correct one is there. If there are only a few solutions, this is a trivial task, but as their number increases, searching through the solution set can become extremely tedious.

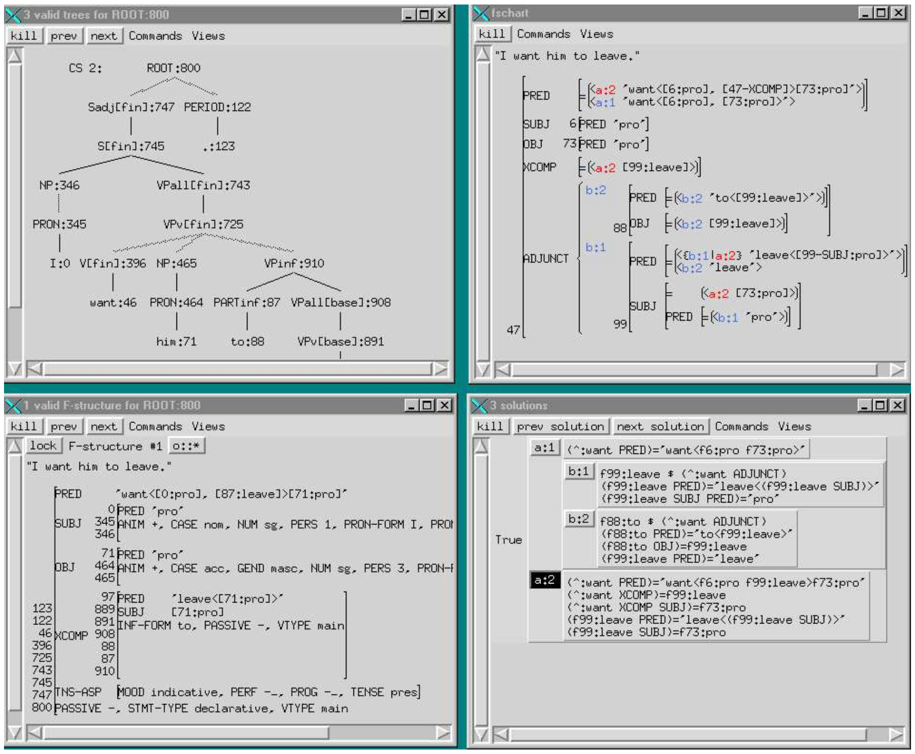
Once the desired analysis is found within the solution set, the task is then to determine the source of the other parses. Some may be legitimate, grammatical analyses, as in the situation described for (5). However, some may indicate overgeneration problems with the grammar which need to be eliminated by the grammar writer. With a large number of solutions, searching through them one at a time can become cumbersome. Having a way to group the solutions can speed up the process. For example, if a lexical item is accidentally entered twice in the lexicon, it can vacuously double the number of parses for any sentence with that lexical item. As such, being able to see this factor of two can speed up the grammar debugging process.

In the remainder of this paper, we discuss various facilities in the XLE system that help the grammar writer manage and view ambiguities in parsing results. The main XLE user interface is shown in Fig. 3. XLE displays four windows: at the top left appears the c-structure window, at the bottom left the corresponding f-structure window. The right-hand side windows display the f-structure chart (on the top) and the chart-choices window (on the bottom). These two views and their usage are discussed in turn below.

4 Displays and Browsing Facilites of Ambiguous Structures

4.1 Manually Searching Through Trees and F-Structures

If there is a relatively small number of trees, then the most obvious way of examining all the parses of a given sentence is to browse through the structures manually, one at a time. For example, the grammar writer may simply want to know whether a given string forms a well-formed VP, independent of the structure of the rest of the sentence, something which can be easily detected while rapidly searching through the tree structures. Within XLE, this type of search is done by clicking on the `next` button in the tree window. In addition to displaying the next tree, clicking the `next` button automatically displays the corresponding f-structure in the f-structure window. In case a single tree



```
CS 2:        ROOT:800

     Sadj[fin]:747  PERIOD:122

       S[fin]:745      .:123

NP:346            VPall[fin]:743

PRON:345           VPv[fin]:725

  I:0  V[fin]:396  NP:465          VPinf:910

        want:46   PRON:464  PARTinf:87  VPall[base]:908

                  him:71      to:88     VPv[base]:891
```

```
"I want him to leave."

    PRED     [<a:2 'want<[6:pro], [47-XCOMP]>[73:pro]'>]
             [<a:1 'want<[6:pro], [73:pro]>'>]
    SUBJ    6[PRED 'pro']
    OBJ    73[PRED 'pro']
    XCOMP    [<a:2 [99:leave]>]
             b:2       PRED [<b:2 'to<[99:leave]>'>]
                    88 OBJ  [<b:2 [99:leave]>]
    ADJUNCT  b:1       PRED [<{b:1|a:2} 'leave<[99-SUBJ:pro]>'>]
                            [<b:2 'leave'>]
                       SUBJ [      <a:2 [73:pro]>]
                    99      PRED [<b:1 'pro'>]
 47
```

```
"I want him to leave."

     PRED     'want<[0:pro], [87:leave]>[71:pro]'
             0 PRED 'pro'
     SUBJ  345 ANIM +, CASE nom, NUM sg, PERS 1, PRON-FORM I, PRO
           346
            71 PRED 'pro'
     OBJ   464 ANIM +, CASE acc, GEND masc, NUM sg, PERS 3, PRON-
           465
            97 PRED    'leave<[71:pro]>'
 123       889 SUBJ    [71:pro]
 122       891 INF-FORM to, PASSIVE -, VTYPE main
  46 XCOMP 908
 396        88
 725        87
 743       910
 745
 747 TNS-ASP  MOOD indicative, PERF -_, PROG -_, TENSE pres
 800 PASSIVE -, STMT-TYPE declarative, VTYPE main
```

```
      a:1  (^:want PRED)='want<f6:pro f73:pro>'
           b:1  f99:leave $ (^:want ADJUNCT)
                (f99:leave PRED)='leave<(f99:leave SUBJ)>'
                (f99:leave SUBJ PRED)='pro'
           b:2  f88:to $ (^:want ADJUNCT)
True            (f88:to PRED)='to<f99:leave>'
                (f88:to OBJ)=f99:leave
                (f99:leave PRED)='leave'
      a:2  (^:want PRED)='want<f6:pro f99:leave>f73:pro'
           (^:want XCOMP)=f99:leave
           (^:want XCOMP SUBJ)=f73:pro
           (f99:leave PRED)='leave<(f99:leave SUBJ)>'
           (f99:leave SUBJ)=f73:pro
```

Figure 3: XLE Windows

is associated with multiple f-structures, these can be viewed by clicking the `next` button in the f-structure window. Fig. 4 shows the second of three trees for the sentence *I want him to leave.* The f-structure corresponding to this particular tree is shown as well (Fig. 5).

Once the grammar writer locates a particular tree, there are a number of facilities for exploring how the subtrees relate to the grammar specification, to the f-structure, and to alternative subtrees in the chart. Clicking on a tree node with different buttons gives three displays. First, the feature constraints specified in the grammar for that category can be examined. Second, the part of the f-structure corresponding to the category can be displayed; this is particularly useful in determining where a given feature in the f-structure comes from. This is demonstrated by Fig. 6 which shows the f-structure corresponding to *I* in the sentence *I want him to leave*; 7 shows the grammar constraints on this f-structure, namely that it not have a SPEC(ifier) and not have an relative clause adjunct. Since these two tools work on subtrees, they can be used to determine where in a tree the f-structure becomes ungrammatical if the structure was intended to be well-formed but was not (or why it is unfortuitously grammatical). Third, alternative subtrees in the chart with the same category can be displayed (the fact that there are alternative subtrees available in the chart is signalled by the dotted lines, as in the VPv tree in Fig. 4). This last feature can be used when the tree that the grammar writer is looking for is not in the set of grammatical trees; the grammar writer can look at the subtrees and from them determine whether the desired tree is present and, if so, why it did not surface.

These facilities are valuable for a focused exploration of a mildly ambiguous string. However, as the trees and f-structures are viewed one at a time, even when the grammar writer knows what to look for, it can become difficult to keep track of the differences between the trees and whether, in fact, there is a non-vacuous ambiguity being captured. Below we discuss how the process of searching through

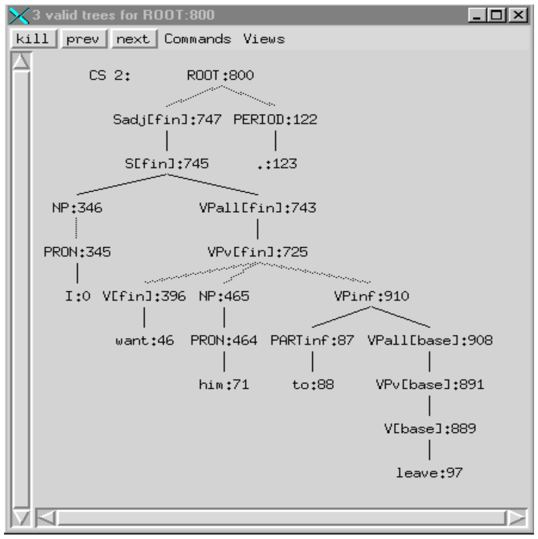
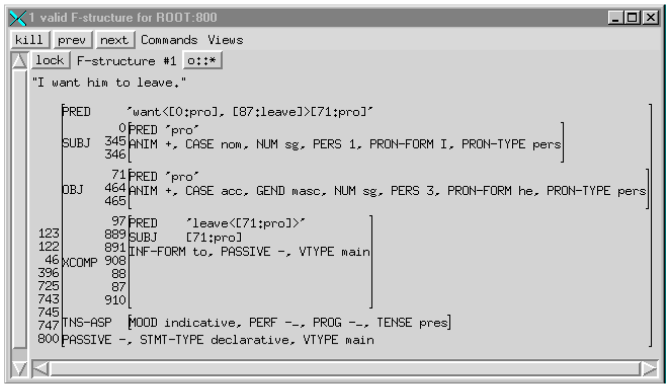


Figure 4: Tree for *I want him to leave.*



```
"I want him to leave."

    PRED     'want<[0:pro], [87:leave]>[71:pro]'
            0 PRED 'pro'
    SUBJ  345 ANIM +, CASE nom, NUM sg, PERS 1, PRON-FORM I, PRON-TYPE pers
          346
           71 PRED 'pro'
    OBJ   464 ANIM +, CASE acc, GEND masc, NUM sg, PERS 3, PRON-FORM he, PRON-TYPE pers
          465
           97 PRED    'leave<[71:pro]>'
123       889 SUBJ    [71:pro]
122       891 INF-FORM to, PASSIVE -, VTYPE main
 46 XCOMP 908
396        88
725        87
743       910
745
747 TNS-ASP   MOOD indicative, PERF -_, PROG -_, TENSE pres
800 PASSIVE -, STMT-TYPE declarative, VTYPE main
```

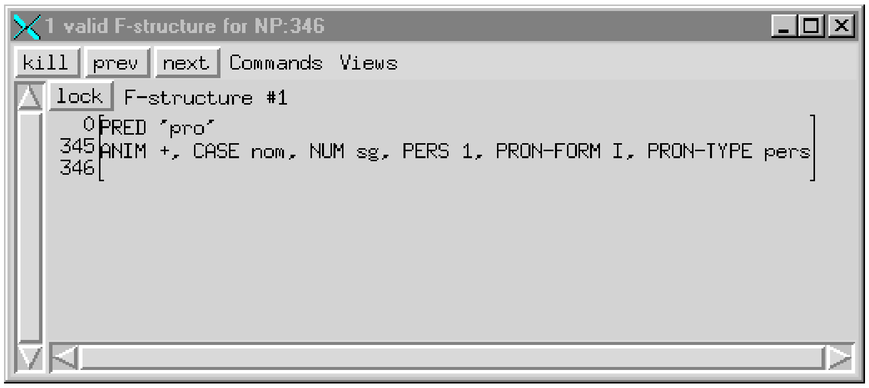
Figure 5: F-structure for *I want him to leave.*

analyses based on their tree-structures can be sped up by specification of parts of the constitutent structure via the bracketing window.

4.2 Sorting by C-Structure Constraints: the Bracketing Window

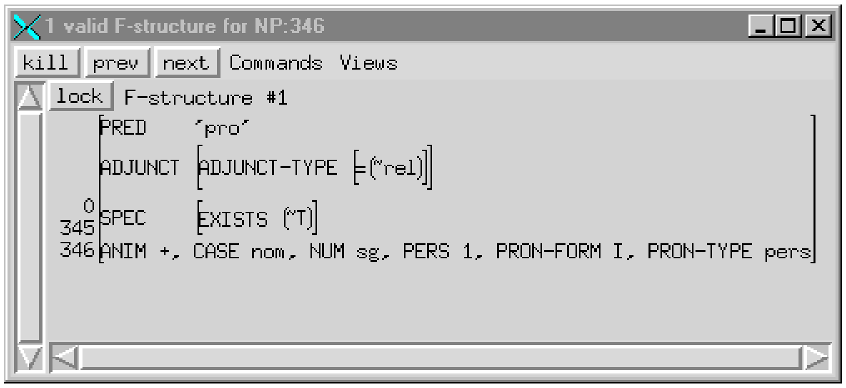
If the grammar writer is looking for a specific analysis of a sentence among many solutions, being able to specify what parts of the tree look like can help immensely in locating the desired tree and corresponding f-structure(s). XLE provides a sophisticated tool which allows the grammar writer to specify constituents, with or without specific labels, and non-constituents. This tool is referred to as the bracketing window. The bracketing window is accessed from the tree window. It allows the grammar writer to systematically narrow down the set of structures displayed, by imposing constraints on a subset of c-structures to be selected from the chart.

In the bracketing window, the sentence is displayed with alternate tokenizations shown above one another. Active buttons (#) appear as token delimitors. Clicking on a pair of these buttons inserts a pair of brackets that encloses the material in between. This imposes a filter on the trees to be



```
  0 PRED 'pro'
345 ANIM +, CASE nom, NUM sg, PERS 1, PRON-FORM I, PRON-TYPE pers
346
```

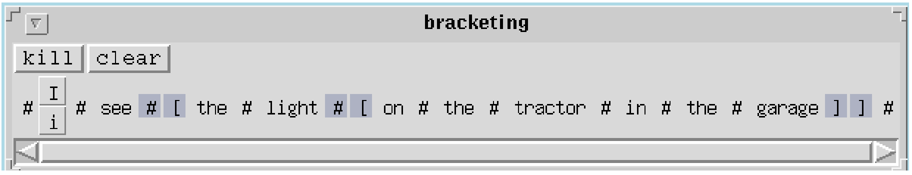
Figure 6: F-structure for *I*



```
    PRED     'pro'

    ADJUNCT  [ADJUNCT-TYPE =c(~rel)]

  0 SPEC     [EXISTS (~T)]
345
346 ANIM +, CASE nom, NUM sg, PERS 1, PRON-FORM I, PRON-TYPE pers
```

Figure 7: F-structure for *I* with grammar constraints

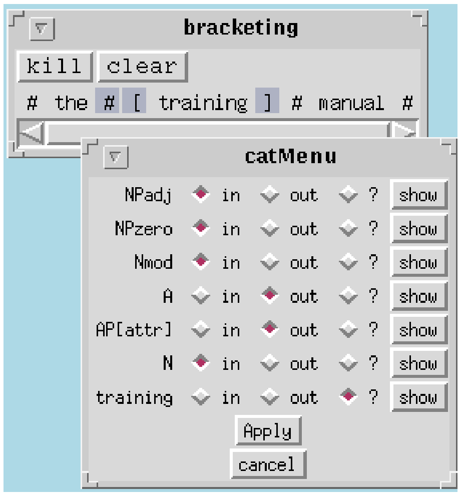
selected from the chart: only those trees and solutions in which the material between brackets forms a constituent are then displayed in XLE's tree and f-structure windows. Alternatively, shift clicking on a pair of these buttons "debrackets" the enclosed material; only those trees and solutions in which the debracketed material does not form a constituent are displayed. This is shown in Fig. 8 in which *the light on the tractor in the garage* and *on the tractor in the garage* must form consituents. There is only one valid tree corresponding to these requirements: the one in which *on the tractor* is an ADJUNCT of *light* and *in the garage* is an ADJUNCT of *tractor*.



```
# I/i # see # [ the # light # [ on # the # tractor # in # the # garage ] ] #
```

Figure 8: Bracket Window for *I see the light on the tractor in the garage*

Clicking on one of the inserted brackets produces a menu of the categories of constituents that span the bracketed material in the chart. These can be specified individually as being included (selected), excluded, or undecided. If a category is specified to be included, only those trees will be displayed in which the constituent that spans the bracketed material is of the chosen category. Conversely, if a category is excluded, XLE filters all trees in which this category spans the bracketed material from the display. Clicking one of the show buttons displays the subtrees in the chart which are labelled with the respective category, to help the grammar writer specify the appropriate category selections. This is shown in Fig. 9 in which the part of speech of *training* is specified to be nominal, by requiring all the noun related categories to be in, and not adjectival, by requiring all the adjective related categories to be out.



the # [training] # manual

NPadj	[✓ in]	out	?	show
NPzero	[✓ in]	out	?	show
Nmod	[✓ in]	out	?	show
A	in	[✓ out]	?	show
AP[attr]	in	[✓ out]	?	show
N	[✓ in]	out	?	show
training	in	out	[✓ ?]	show

Figure 9: Bracket Window for *the training manual*

A few bracketing constraints often considerably narrow down the search space, such that the browsing method described above is applicable even with highly ambiguous sentences. Narrowing down the search space by imposing c-structure constraints is also often the first step before application of the more sophisticated selection strategies to which we turn next.

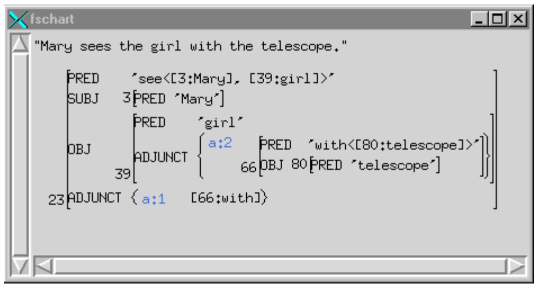
The bracketing tool has proven particularly useful for treebanking tasks where the tree banker often has a solid intuition as to what the desired tree should be. Using the bracketing window to guarantee the correct constituency of the tree, especially when combined with specifying categories (e.g., noun vs. adjective), greatly speeds up the treebanking process. An additional advantage of the bracketing window for treebanking is that tree bankers who are unfamiliar with the grammar and even with the LFG formalism can use it to quickly choose the correct solution for a given sentence since all that is required is a knowledge of constiuency. For similar reasons, this type of tool should be useful for any grammar which produces tree structures as part of its output.

4.3 Sorting by F-Structure Context Variables

Although searching through c-structure trees, with or without the aid of the bracketing window, is ideal for certain applications, for grammar testing it is often necessary to check the f-structure space since the details of the LFG syntactic analyses are located here (e.g., verb subcategorization information). As mentioned in section 2.2, XLE provides a display of the packed f-structure representations used in the parsing and generation algorithms. The logical context variables employed in the contexted conjunctive form appear as choices in the display, labelled *a:1, a:2, a:3, . . . b:1, b:2, b:3*, etc. This type of display allows the grammar writer to view all of the f-structures for a sentence at one time. With some experience, the compact representations of the entire solution space are very useful to the grammar writer, for determining both whether a desired f-structure is present and whether there is any unexpected overgeneration by the grammar. Since the choices among the different f-structures in the displays are active, they allow quick access to individual readings based on two different indexing criteria which are discussed in sections 4.3.1 and 4.3.2. When the grammar writer clicks on a choice, the solution corresponding to that choice is displayed in the tree and f-structure windows. A selection is a fully specified choice of exactly one solution.

4.3.1 Packed F-Structure

The f-structure chart window indexes the packed solutions by their constraints, so that each constraint appears once in an f-structure annotated by all of the choices where that constraint holds. This is best seen in the f-structure chart in Fig. 1 above, repeated as Fig. 10: the f-structure for the PP *with the telescope* appears only once, although it can attach either high, appearing as an adjunct to the verb's f-structure (*a:1*), or low, appearing in the f-structure of *girl* (*a:2*).



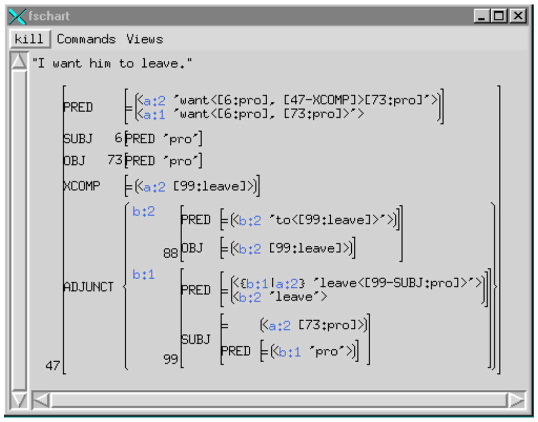
```
"Mary sees the girl with the telescope."

   [PRED    'see<[3:Mary], [39:girl]>'                              ]
   |SUBJ   3[PRED 'Mary']                                           |
   |          [PRED    'girl'                                      ]|
   |OBJ       |         { a:2   [PRED  'with<[80:telescope]>'  ]} ||
   |          |ADJUNCT  {     66[OBJ 80[PRED 'telescope']      ]} ||
   |        39[                                                    ]|
 23[ADJUNCT { a:1   [66:with]}                                      ]
```

Figure 10: F-structure chart for *Mary sees the girl with the telescope.*

In Fig. 11, the f-structure chart for *I want him to leave* is seen. The *a:1/a:2* choice reflects the separate argument frames of *want*. The *b:1/b:2* choice has a number of correlated effects on the structure (some of which are not shown in this "preds only" view, see below): in *b:1*, we have the noun *leave* acting as the object of the preposition *to*; in *b:2*, *leave* is an infinitival adjunct, with its subject 'pro' being present only at the level of f-structure. The choice labels are color-coded: one consistent selection of choices is highlighted in red (e.g., *a:2, b:1*), all other choice labels are blue.[6] (The selected reading is simultaneously displayed in the non-packed tree and f-structure windows discussed in section 4.1.) Clicking on a non-selected choice will change the selected reading, with all dependent choices being adjusted in a way that results in a consistent overall selection of choices.

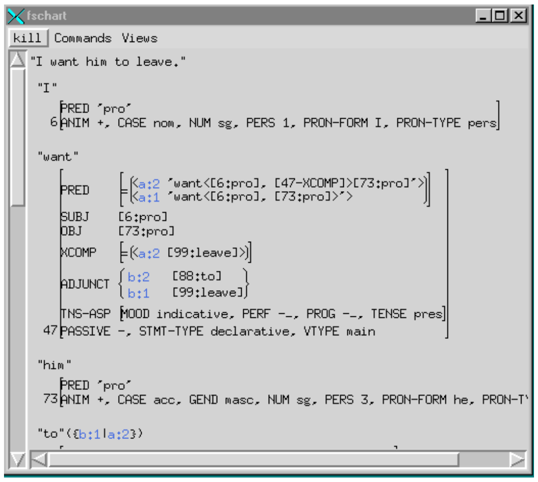
F-structures contain a large amount of information, and for many longer sentences this results in structures, especially packed structures, which cannot be easily displayed on the screen. To minimize this problem, there are various ways to control how the f-structure is displayed. For example, the "preds only" menu item suppresses all of the attributes except PRED, the governable attributes, and the semantic attributes. This display is very useful to the grammar writer when searching for particular predicate argument relations and when making a first estimate as to whether two analyses are identical or not. Another option is the "linear" menu item which changes the display into a line of surface forms with corresponding f-structures. The linear display is very useful with large f-structures and multiple dependent context choices since it gives the grammar writer natural and quick access to local disjunctions, in that they are indexed on the individual words of the string. The first part of the linear display for *I want him to leave.* is shown in Fig. 12.

[6]Besides the red buttons, which select a single solution, XLE also provides grey buttons, which only narrow down the solution space without immediate selection of a single solution.



```
"I want him to leave."

   PRED      =(<a:2 'want<[6:pro], [47-XCOMP]>[73:pro]'>)
              (<a:1 'want<[6:pro], [73:pro]>'>)
   SUBJ    6 [PRED 'pro']
   OBJ    73 [PRED 'pro']
   XCOMP     =(<a:2 [99:leave]>)
              b:2      PRED =(<b:2 'to<[99:leave]>'>)
                    88 OBJ  =(<b:2 [99:leave]>)
              b:1      PRED =(<{b:1|a:2} 'leave<[99-SUBJ:pro]>'>)
   ADJUNCT {                 (<b:2 'leave'>)
                       SUBJ =      (<a:2 [73:pro]>)
                            PRED =(<b:1 'pro'>)
                    99
47
```

Figure 11: F-structure Chart Window for *I want him to leave.*



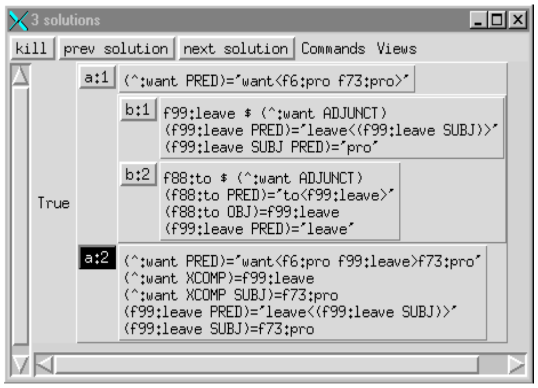
```
"I want him to leave."
 "I"
    PRED 'pro'
  6 ANIM +, CASE nom, NUM sg, PERS 1, PRON-FORM I, PRON-TYPE pers

 "want"
    PRED      =(<a:2 'want<[6:pro], [47-XCOMP]>[73:pro]'>)
               (<a:1 'want<[6:pro], [73:pro]>'>)
    SUBJ      [6:pro]
    OBJ       [73:pro]
    XCOMP     =(<a:2 [99:leave]>)
    ADJUNCT  { b:2   [88:to]    }
             { b:1   [99:leave] }
    TNS-ASP  [MOOD indicative, PERF -_, PROG -_, TENSE pres]
 47 PASSIVE -, STMT-TYPE declarative, VTYPE main

 "him"
    PRED 'pro'
 73 ANIM +, CASE acc, GEND masc, NUM sg, PERS 3, PRON-FORM he, PRON-T

 "to"({b:1|a:2})
```

Figure 12: Linear Display for *I want him to leave.*

4.3.2 Choice Window

The choices displayed in the packed f-structure window are alternatively displayed in the f-structure chart choices window. The choices window indexes the packed solutions by the alternative choices and displays them as a nested tree. The choices that belong to the same disjunction have the same alphabetic string as a prefix. At the left of each disjunction is its context. Top level disjunctions are given the True context. Embedded disjunctions are given the choice that they are embedded under.

The choices window, although not ideal for getting a general feel for the packed f-structure as a whole, is extremely useful for seeing the different ambiguity sources and how they relate to one another. For example, in Fig. 13 it is easy to see that one ambiguity depends on the subcategorization of the verb *want* and that if transitive *want* is chosen (choice *a:1*), then there is an ambiguity as to the role of *leave*, a choice which is not available with the ditransitive form of *want* (choice *a:2*). One particularly interesting consequence of this display is that if two f-structures are vacuously different (e.g., the result of having the same word entered twice in the lexicon), there will be blank lines after the choice labels; when two or more blank lines appear, it is a good indication that something is seriously wrong with the grammar with respect to the given sentence.



```
a:1  (^:want PRED)='want<f6:pro f73:pro>'
  b:1  f99:leave $ (^:want ADJUNCT)
       (f99:leave PRED)='leave<(f99:leave SUBJ)>'
       (f99:leave SUBJ PRED)='pro'
  b:2  f88:to $ (^:want ADJUNCT)
       (f88:to PRED)='to<f99:leave>'
       (f88:to OBJ)=f99:leave
       (f99:leave PRED)='leave'
a:2  (^:want PRED)='want<f6:pro f99:leave>f73:pro'
     (^:want XCOMP)=f99:leave
     (^:want XCOMP SUBJ)=f73:pro
     (f99:leave PRED)='leave<(f99:leave SUBJ)>'
     (f99:leave SUBJ)=f73:pro
```

Figure 13: Choice Window for *I want him to leave.*

5 Print Ambiguity Sources

The above features allow the grammar writer to see the structure of a sentence in detail and to determine the type of ambiguity that is present. However, it is still necessary to pin point the exact source of the ambiguity, e.g., what rule in what file is causing the ambiguity, especially if the grammar writer needs to eliminate the ambiguity in question. XLE provides a feature (print-ambiguity-sources) that prints all of the local sources of ambiguity in the current chart. The output represents both f-structure ambiguities and c-structure ambiguities. The f-structure ambiguities will give a subtree identifier plus the line number of the source of the constraints to help the grammar writer identify the ambiguity source. Whenever a subtree is found that has a large number of local solutions, it is possible to give the subtree identifier to print-ambiguity-sources to find out what ambiguities are being multiplied together to produce the solutions.

(6) shows the result of print-ambiguity-sources for the sentence *I want him to leave*. For example, the first line indicates that V_BASE (the verb stem for *want*) is two way ambiguous and points to the location of the lexical entry for the verb (line 17893 of the file eng-verb-lex.lfg); note that the "perhaps" in (6) is because there is the possibility that the ambiguity is the result of functional uncertainty, which cannot be detected by the tool.

(6) print-ambiguity-sources
V_BASE:47:1 adds a 2-way ambiguity, perhaps from line 17893 in eng-verb-lex.lfg
PRON_SFX_BASE:73:1 adds a 2-way ambiguity, perhaps from line 1674 in eng-core-lex.lfg
VPv:621 is ambiguous because of subtrees 3 & 5 & 6

6 Suppressing Unlikely Analyses

The previous sections described tools which XLE provides for sorting through large numbers of parse results. However, XLE also provides a way to cut down the original search space in parsing, allowing for potentially fewer parses to search through. This is done via a special STOPPOINT feature, which is part of the Optimality Theory preference mechanism incorporated into XLE. It should be possible to incorporate such a system into other parsing systems since this version of Optimality Theory is an overlay on the grammar and not specific to LFG.

Frank et al. (1998, 2000) propose an extension of the classical LFG projection architecture to incorporate a constraint ranking mechanism inspired by Optimality Theory. A new projection, the *o*–projection, is used to specify violable constraints, which are used to determine a "winner" among competing, alternative analyses. In the case of unwarranted syntactic analyses, constraint ranking can often effectively filter such analyses from the output; constraint ranking can also be used to rank competing analyses according to a hierarchy over preference and dispreference constraints. A considerable number of ambiguities can be successfully filtered from the set of possible analyses for a given sentence by using this constraint ranking mechanism, implemented in the XLE system. However, in some cases the "optimal" analyses determined in this way do not correspond to the actual preferred reading of a given sentence, or else, a relatively large set of (equally) optimal analyses may subsist. If the intended analysis does not surface as "optimal", specific optimality marks can still be deactivated after parsing, which results in a correspondingly enlarged solution space.

As an alternative to manual deactivation of specific preference or dispreference marks, special STOPPOINT marks can be inserted into the hierarchy of preference and dispreference marks. If the hierarchy includes one or more instances of the STOPPOINT optimality mark, XLE will process the input in multiple passes, using larger and larger versions of the grammar in subsequent reparsing phases. STOPPOINTs are useful for eliminating ungrammatical analyses when grammatical analyses are present and for speeding up the parser by only using expensive and rare constructions when no other analysis is available.[7]

Optimality marks are processed in right to left order, so that the first STOPPOINT considered is the rightmost. During the first pass, any constructions that are associated with marks to the left of the STOPPOINT are not considered, i.e. not part of the grammar being used. If a solution can be found with this restricted grammar, XLE will terminate with this solution. Otherwise, a reparsing phase is triggered, which will reprocess the input using the grammar up to the next STOPPOINT to the left. For instance, if a grammar had the ranking in (7), then XLE would first try analyses with either no marks or only the Mark1 mark. It would not try the suboptimal constructions involving Mark2 or Mark3. If there were no valid analyses, then it would reparse the sentence, including analyses with a Mark2 mark. If there were still no valid analyses, then it would try including analyses with a Mark3 mark.

(7) OPTIMALITYORDER Mark3 STOPPOINT Mark2 STOPPOINT Mark1.

If one of the marks to the left of a STOPPOINT is a preference mark, then the suboptimal constructions that are not tried are the ones that have fewer preference marks than a competing analysis. Putting a preference mark to the left of a STOPPOINT makes sense for multi-word expressions, for instance. If the preference mark for multi-word expressions is to the left of a STOPPOINT, then XLE will only consider analyses that involve the individual components of the multi-word expression if there is no valid analysis involving the multi-word expression. This is particularly useful for parsing texts with large numbers of multiword technical terms. Reprocessing in multiple passes is expensive, so STOPPOINTs should be used sparingly. The ideal is to have a STOPPOINT at a place that allows 80-90% of the inputs to be processed successfully, and to put the optimality marks of computationally expensive and syntactically marginal grammar rules to the left of a STOPPOINT. With such settings, one can obtain a substantial processing speed-up.

[7]The order in which the ranking of constraint marks is specified has been reversed since Frank et al. (1998, 2000). It now starts with the worst constraint to be violated, reflecting the order assumed in theoretical work within Optimality Theory.

7 Conclusion and Discussion

The ambiguity management tools described here are not only used during grammar development, but also in tasks requiring disambiguation by a human, such as treebanking or the design of evaluation test suites (which can be seen as a special case of treebanking). These tools are especially convenient for such tasks since they allow non-expert users to locate specific solutions among the output set more straightforwardly.

When creating a treebank, the task is to save the correct tree and corresponding f-structure analysis for each sentence in the corpus. As treebanking often involves long, naturally occurring sentences, each sentence can have a large number of parses (tens and even hundreds). Given the number of parses and the number of sentences to bank, having a way of efficently zeroing in on the correct parse is vital to the task. The tools described in this paper have been tested on two relatively small (hundreds instead of thousands of sentence) treebanks created for this project for English, and for a parallel corpus in French. The difficulties encountered in these tasks were instrumental in guiding the current state of the tools. Currently, the tools are being used in a much larger tree banking project for German (Dipper (2000)).

In this paper we reported on various tools in the XLE grammar development platform which can be used for ambiguity management in grammar writing. In particular, we looked at packed representations of ambiguities that allow the grammar writer to view sorted descriptions of ambiguity sources, as well as tools for specifying desired tree structures and for cutting down the solution space prior to parsing. These tools allow the grammar writer to create large-scale linguistically motivated grammars and apply them to real-life corpora, while keeping track of ambiguity sources. It is our hope that these basic ideas behind these tools are fundamental enough to prove useful to grammar writers using other frameworks or platforms.

References

Bresnan, J., editor (1982). *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, MA.

Bresnan, J. (2000). *Lexical-Functional Syntax*. Blackwell Publishers. To appear. Ms., Stanford University.

Butt, M., King, T. H., Niño, M.-E., and Segond, F. (1999). *A Grammar Writer's Cookbook*. CSLI Publications, Stanford, CA.

Carter, D. (1997). The TreeBanker: A tool for supervised training of parsed corpora. ACL Workshop: Computational Environments for Grammar Development and Linguistic Engineering, Madrid.

Dipper, S. (2000). Grammar-based corpus annotation. Presented at the LINC-2000 Workshop, Luxembourg.

Frank, A., King, T., Kuhn, J., and Maxwell, J. (1998). Optimality theory style constraint ranking in large-scale lfg grammars. In Butt, M. and King, T., editors, *Proceedings of the LFG98 Conference*, University of Queensland, Brisbane, CSLI Online Publications, Stanford, CA. `http://www-csli.stanford.edu/publications/`.

Frank, A., King, T. H., Kuhn, J., and Maxwell, J. (2000). Optimality theory style constraint ranking in large-scale LFG grammars. In Sells, P., editor, *Formal and Empirical Issues in Optimality Theoretic Syntax*. CSLI Publications, Stanford, CA. To appear.

Maxwell, J. and Kaplan, R. (1989). An overview of disjunctive constraint satisfaction. In *Proceedings of the International Workshop on Parsing Technologies*, pages 18–27. also published as: A method for disjunctive constraint satisfaction. in: Tomita, M. (ed): *Current Issues in Parsing Technology*, Kluwer Academic Publishers, 1991).

Maxwell, J. and Kaplan, R. (1993). The interface between phrasal and functional constraints. *Computational Linguistics*, 19(4):571–590.

Maxwell, J. and Kaplan, R. (1996). Unification-based parsers that automatically take advantage of context freeness. Paper presented at the *LFG96 Conference*, Grenoble, France. Ms. Xerox PARC.

Shemtov, H. (1997). *Ambiguity Management in Natural Language Generation*. PhD thesis, Stanford University.