Research on Language and Computation **2:** 259–280, 2004.

Ambiguity Management in Grammar Writing

TRACY HOLLOWAY KING[1], STEFANIE DIPPER[2], ANETTE FRANK[3], JONAS KUHN[4] and JOHN T. MAXWELL III[5]
[1]*NLTT/ISTL, PARC, 3333 Coyote Hill Rd, Palo Alto, CA 94304 USA (E-mail: thking@parc.com);* [2]*Institut für maschinelle Sprachverarbeitung, Universität Stuttgart, Azenbergstraße 12, 70174 Stuttgart, Germany (E-mail: dipper@ims.uni-stuttgart.de);* [3]*DFKI, Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany (E-mail: frank@dfki.de);* [4]*Department of Linguistics, 1 University Station, B5100, University of Texas at Austin, Austin, TX 78712 USA (E-mail: jonask@mait.utexas.edu);* [5]*NLTT/ISTL, PARC, 3333 Coyote Hill Rd, Palo Alto, CA 94304 USA (E-mail: maxwell@parc.com)*

Abstract. When linguistically 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.

Key words: ambiguity management, grammar writing, large-scale grammars, XLE, packed representations

1. Introduction

With the interest in broad coverage grammars and the number of available systems growing, the need for well designed grammar writing tools has become more and more obvious. In many papers about grammar development this issue is mentioned – however, most of those papers focus on parsing techniques. Detailed descriptions of grammar writing tools are typically restricted to user manuals. Since manuals have to cover all options of the tool, they cannot address special aspects of the tools and appropriate grammar development techniques in depth.

One aspect of grammar engineering, namely systematic profiling of grammar coverage by use of test suites and support by special test suite analysis tools, is relatively well covered in the literature (cf. Lehmann et al., 1996; Kuhn, 1998; Oepen and Callmeier, 2000). Since all large-scale grammar development systems provide for some more or less sophisticated test suite analysis tools for general coverage, performance measuring, and regression testing, we will not focus on this aspect in the present paper.

What standard test suite analysis tools can provide only to a limited degree is the focussed control of ambiguity from the grammar writer's perspective. It

is this specific, complex task in practical large-scale grammar development that we focus on in this paper. We illustrate and discuss the techniques and tools employed in this task based on a particular grammar development platform, the Xerox Linguistic Environment (XLE). XLE is a platform designed specifically for the development of grammars in the framework of Lexical-Functional Grammar (LFG). However, most of the engineering aspects of large-scale grammar carry over to other frameworks and development platforms.

The paper is organized as follows: Section 2 discusses the different facets of the ambiguity problem in computational syntax, identifying the problem of ambiguity management as the central issue in large-scale grammar development. Section 3 introduces the specific project context that we use for illustration, presenting the main interfaces of the XLE system in section 3.1 and providing some background on the technical approach to ambiguity packing implemented in the XLE system in section 3.2. Section 4 is the main section of this paper, discussing the tools and techniques for ambiguity management in XLE; in sections 4.1 and 4.2 we present the basic facilities for viewing and selecting analyses; 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; section 4.4 presents an additional device for detecting ambiguity sources in a grammar. In section 5, we compare the tools that the XLE system provides with the facilities of a related approach. Finally, section 6 provides a summary with some discussion.

2. Ambiguity Management in Grammar Development

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 three main aspects of managing ambiguity in language processing: (i) The parsing and generation algorithms have to deal with an exponential number of ambiguous structures in an efficient way, avoiding combinatorial explosion. (ii) Some disambiguation strategy has to be adopted, depending on the application of the grammar (one strategy is 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 (see for example Maxwell and Kaplan 1989, 1993, 1996; Shemtov, 1997; Oepen and Carroll 2000; Flickinger, 2000).

However there is a third aspect of the ambiguity problem that is independent of these two: (iii) 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. This aspect is what we will focus on.

2.1. DEALING WITH AMBIGUITY IN GRAMMAR WRITING – AN EXAMPLE

We can distinguish two major ambiguity-related tasks for 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 (1a). Running a relatively large-scale grammar, a linguist will typically expect three analyses for this sentence (1b–d):

(1) a. I saw her duck under the table.

b. I saw [$_{NP}$ her] [$_{VP}$ duck under the table].
= I saw that she ducked under the table.

c. I [$_{VP}$ saw [$_{NP}$ her duck] [$_{PP}$ under the table]].
(the seeing is done under the table)

d. I saw [$_{NP}$ her duck [$_{PP}$ under the table]].
(the duck is under the table)

As a matter of fact, a grammar based on a realistic lexicon will reveal two additional analyses with *saw* as present tense of the transitive verb *saw* (with the same attachment possibilities as in (1c, d)). On purely syntactic grounds these readings are fully justified, and their presence underlines the need for ambiguity management tools. In the following, however, we will restrict attention to the three readings in (1b–d), with the verb *see*. The first analysis (1b) is one in which *see* takes three arguments (subject, object, and infinitive); in the other two it takes two arguments (subject and object) and an adjunct to the verb or to the object.

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 (1). 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 for a lexical item an extra, incorrect subcategorization frame has been added to the grammar, the number of parses for any sentence with that lexical item will grow systematically. As such, being able to see this increase in the number of readings can speed up the grammar debugging process.

It should be evident that for ambiguity management in large-scale grammars, various grammar debugging approaches should be combined: in some cases, a detailed inspection of individual syntactic analyses will reveal problems or bugs in the grammar. However, in order to efficiently find out which input sentences should be inspected after a revision in the grammar, suitable test suites and an automated test regime with profiling techniques is required. As mentioned above, test suite design issues have been extensively discussed in the literature (see Lehmann et al., 1996 and references therein), and test suite analysis tools are provided with all grammar development platforms (see for example Kuhn, 1998; Oepen and Callmeier, 2000). We therefore do not address testing in the present paper, but focus on the tools and techniques required for inspecting ambiguous analyses for individual input sentences, possibly detected by using general test suite analysis tools.

2.2. AMBIGUITY MANAGEMENT TOOLS IN GRAMMAR DEVELOPMENT SYSTEMS

All current systems for grammar development on a large scale contain tools supporting the grammar writer. Many of them provide facilities similar to those we present in the following section. However, for reasons mentioned in the introduction, specialized reports about tools supporting ambiguity management are missing (with the exception of Carter (1997), discussed in more detail below).

Here is an overview of many of the more widely known large grammar development systems (focussing on publications about grammar writing facilities).

- The Grammar Development Environment (GDE) is part of the Alvey Natural Language Tools. It employs a metagrammatical formalism similar to GPSG. The GDE user manual (Carroll, 1991) includes descriptions of various tool commands for parsing and debugging.
- The Linguistic Knowledge Building system (LKB) is a grammar and lexicon development environment designed for constraint-based linguistic formalisms (Copestake and Flickinger, 2000). The LKB user manual (Copestake et al., 2000) also includes descriptions of various tool commands for parsing and debugging.
- The XTAG system includes a grammar development tool based on the Tree Adjoining Grammar formalism. Short descriptions of the tool can be found in XTAG Research Group (2001), chapter 3, and Doran et al. (1994) (note that in the 2001 version of XTAG Research Group, this paragraph is missing).

- PAGE (Platform for Advanced Grammar Engineering) is a development platform of grammars based on typed feature logics (e.g. HPSG) with limited documentation.
- There are various grammar development environments designed with the predominant goal of verifying linguistic theories: ALE (Carpenter and Penn 1999), CUF (Dörre et al., 1996), TFS (Emele, 1994), ConTroll (Götz et al., 1997), XLFG (Clement, 1996–1999), and others.[1] With such systems the need for sophisticated inspection tools is less immediate since they are mostly used with smaller grammar fragments.

As opposed to the references cited for the systems listed above, Carter (1997) is a detailed report about a tool used for disambiguation within the Core Language Engine (CLE) (Alshawi, 1992). CLE is a general purpose system for natural language applications. It includes a syntactic parsing component for constraint-based unification grammars, and delivers semantic representations in QLF (Quasi-Logical Form) format. Carter (1997) presents a well-designed graphical tool, the "TreeBanker", which enables linguistically trained, but non-expert users to efficiently select analyses from a set of proposed solutions delivered by the CLE system. Differences between the TreeBanker tools and the facilities provided by the XLE system will be discussed in section 5.

3. The Grammar Development Platform

This section provides the relevant background on the XLE system, focussing on the technical treatment of ambiguity in parsing and giving a first overview of the display facilities.

3.1. PROJECT CONTEXT

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: originally, 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). The c-structure encodes

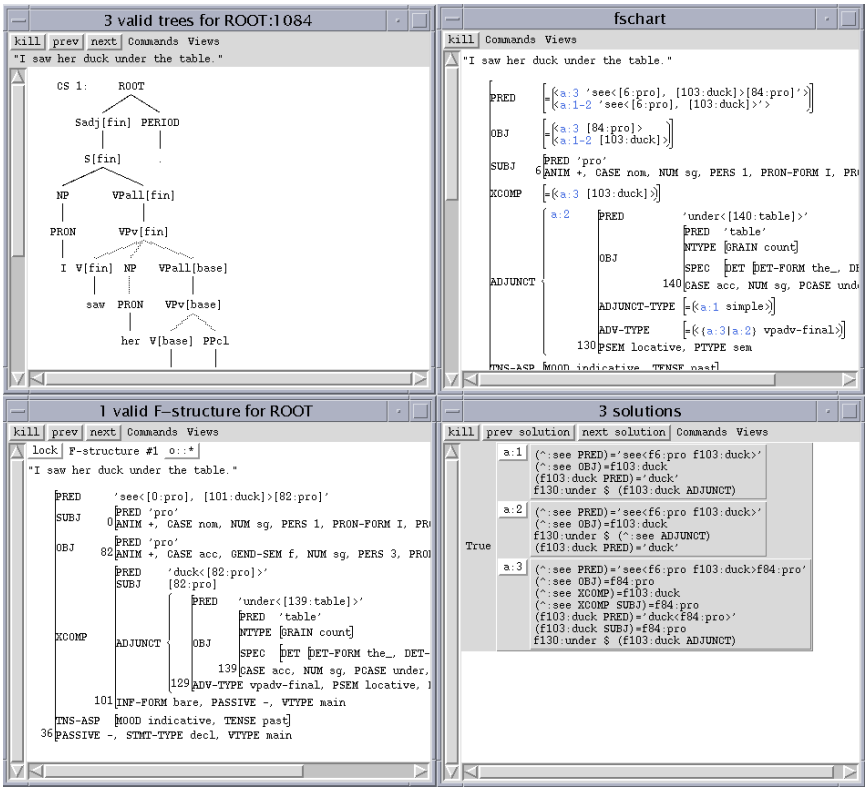

Figure 1. XLE Windows.

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).

The main XLE user interface is shown in Figure 1. 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 views and their usage are discussed in detail in section 4.

3.2. AMBIGUITY PACKING IN XLE

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; Shemtov, 1997).[2]

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 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 (2). 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.

(2)
$$\begin{array}{lll} & & (a \wedge x \wedge c) \\ & \text{DNF} & \vee\ (a \wedge x \wedge d) \\ (a \vee b) \wedge x \wedge (c \vee d) & \Rightarrow & \vee\ (b \wedge x \wedge c) \\ & & \vee\ (b \wedge x \wedge d) \end{array}$$

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 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 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 $(a, b, x, \ldots)$ is labeled with a propositional (context) variable p, q or its negation, as in (3)

(3)
$$\begin{array}{lll} & \text{CF} & (p \rightarrow a) \wedge (\neg p \rightarrow b) \\ (a \vee b) \wedge x \wedge (c \vee d) & \Rightarrow & \wedge\ x \\ & & \wedge\ (q \rightarrow c) \wedge (\neg q \rightarrow d) \end{array}$$

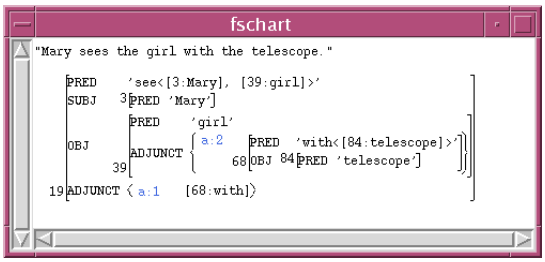

Figure 2. F-structure chart for *Mary sees the girl with the telescope*.

based on the Lemma in (4):

(4) $\phi_1 \vee \phi_2$ is satisfiable iff
$(p \rightarrow \phi_1) \wedge (\neg p \rightarrow \phi_2)$ 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 $\phi_1 \vee \phi_2$ at least one of the disjuncts is true.

As can be seen in the above example (3), 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 also allows for the *representation* of a set of ambiguous f-structures in a single, packed f-structure, the so-called f-structure chart, where disjunctive facts are not compiled out and duplicated. Very similar to what we see in formula (3), in the packed f-structure chart, attribute-values (facts) are indexed with their corresponding context variables, as displayed in Figure 2 for the sentence (5).

(5) Mary sees the girl with the telescope.

The f-structure in Figure 2 represents context variables by labels *a:1* and *a:2*. Additional numbers in the f-structure (19, 3, 39, 68) name f-structure nodes (usually referred to as f_{19}, f_3, etc). This f-structure shows only the PRED values and no other attributes. For a more complete f-structure, see Figure 5.

In (5), the ambiguity resides in the attachment level of the PP as a VP- or NP-adjunct, as seen graphically in the trees in Figure 3. While this ambiguity affects the entire c-to-f-structure mapping from the level of VP down, it is captured by the single local disjunctive contexts *a:1* and *a:2* in the f-structure chart in Figure 2. Here, context *a:1* specifies the PP identified by f-structure node f_{68} as an element of the ADJUNCT set in the main predicate's f-structure f_{19}. In context *a:2*, the PP (f_{68}) is specified as an element of the OBJect's (f_{39}) ADJUNCT set. All remaining

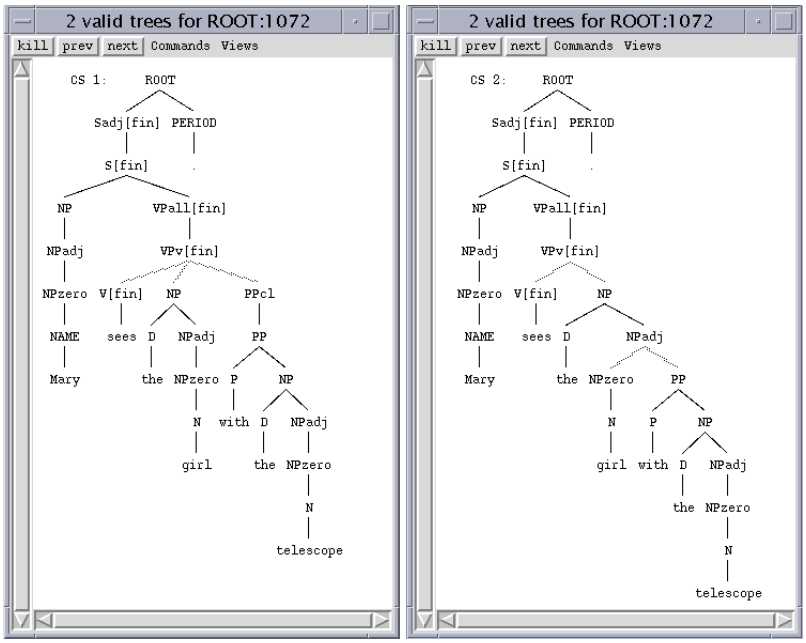

Figure 3. Trees for *Mary sees the girl with the telescope*.

f-structure constraints are conjoined in the TRUE context (the TRUE variable is the default context variable and is not displayed in Figure 2).

In other words, the f-structure chart represents the disjunctive contexted facts *a:1*: $f_{68} \in (f_{19}\ \text{ADJUNCT})$ (the PP's f-structure is an adjunct of the verb) and *a:2*: $f_{68} \in (f_{39}\ \text{ADJUNCT})$ (the PP is an adjunct of OBJ), along with all remaining f-structure constraints, which are true of both analyses. (6) redisplays (part of) the contexted, implicational constraints of Figure 2 in the notation used in (4) above: <context variable> $\rightarrow$ <fact>.

$$
\begin{array}{lll}
(6) & a{:}1 & \rightarrow f_{68} \in (f_{19}\ \text{ADJUNCT}) \\
\wedge & a{:}2 & \rightarrow f_{68} \in (f_{39}\ \text{ADJUNCT}) \\
\wedge & \text{TRUE} & \rightarrow (f_{19}\ \text{OBJ}) = f_{39} \\
\wedge & \text{TRUE} & \rightarrow (f_{19}\ \text{PRED}) = \text{`see}\langle f_3,\ f_{39}\rangle\text{'} \ldots
\end{array}
$$

In the packed f-structure view of Figure 2, the local disjuncts are directly accessible through their context variables. That is, we can select a reading from the chart by selecting (clicking) its corresponding context variable. Clicking context variable *a:1* implicitly sets it to the TRUE context, while the alternative context *a:2* is set to FALSE.[3] We thus select the analysis in which the PP is an adjunct of the OBJ. The resulting contexted constraint system is displayed in (7).

$$
\begin{array}{lll}
(7) & \text{TRUE} & \rightarrow f_{68} \in (f_{19}\ \text{ADJUNCT}) \\
\wedge & \text{FALSE} & \rightarrow f_{68} \in (f_{39}\ \text{ADJUNCT}) \\
\wedge & \text{TRUE} & \rightarrow (f_{19}\ \text{OBJ}) = f_{39} \\
\wedge & \text{TRUE} & \rightarrow (f_{19}\ \text{PRED}) = \text{`see}\langle f_3,\ f_{39}\rangle\text{'} \ldots
\end{array}
$$

Section 4.3 will describe in more detail how different displays of packed f-structure representations support grammar writers in efficiently filtering and selecting solutions from a single view on the full set of alternative f-structures.

4. Displays and Browsing Facilities of Ambiguous Structures

In this section, we discuss tools for searching through trees and functional structures to locate both desired analyses and unwanted ambiguities. Although the tools were designed in XLE for an LFG grammar, the c-structure based tools should be relevant to any grammar that uses tree structures, while the f-structure based tools should be relevant for grammars using attribute-value matrices and more generally for ambiguities involving grammatical functions.

4.1. MANUALLY SEARCHING THROUGH TREES AND F-STRUCTURES

If there is a relatively small number of trees, then the most obvious way to examine 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 is associated with multiple f-structures, these can be viewed by clicking the `next` button in the f-structure window. Figure 4 shows the first of three trees for the sentence *I saw her duck under the table*. The f-structure corresponding to this particular tree is shown as well (Figure 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 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 Figure 7 which shows the f-structure corresponding to *I* in the sentence *I saw her duck under the table*.

Second, the feature constraints specified in the grammar for that category can be examined. Figure 7 shows the grammar constraints on the f-structure corresponding to the head noun of the NP *the table*. In this case, it shows the expansion

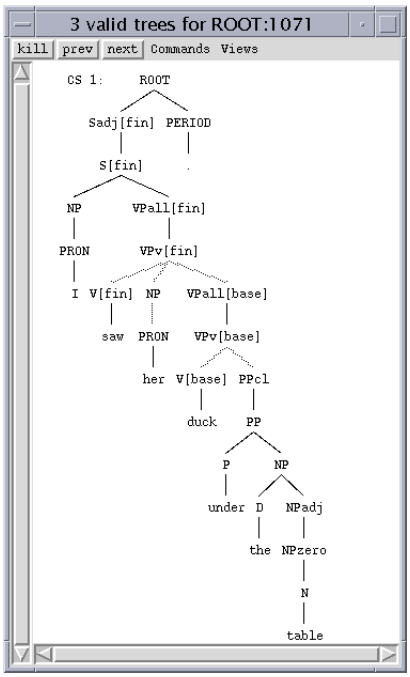

Figure 4. Tree for *I saw her duck under the table*.

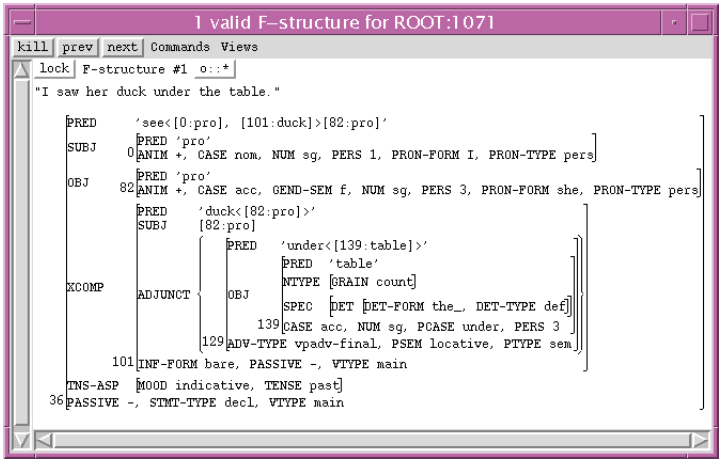

Figure 5. F-structure for *I saw her duck under the table*.

of the templates called by *table*: it is a noun with a simple PRED or with a SUBJ (for predicative position), and as a count noun it must have a SPECifier if singular. This information can be used to determine what the grammar requirements are on the particular part of the tree, in this case just the head noun of the NP. Since these two tools work on subtrees, they can be used to determine where in a tree the f-

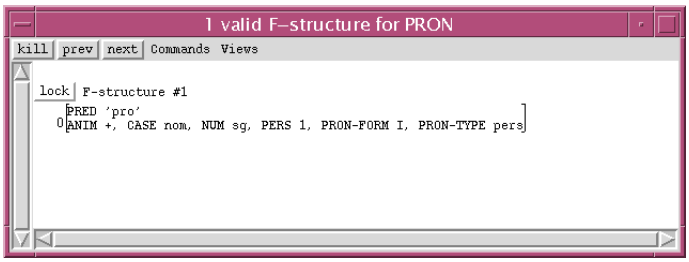

Figure 6. F-structure for *I*.

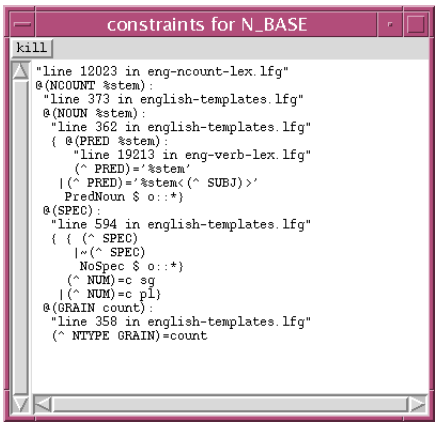

Figure 7. Grammar constraints for *table*.

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[fin] tree in Figure 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 in the chart 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 analyses based on their tree-structures can be sped up by specification of parts of the constitutent structure via the bracketing window.

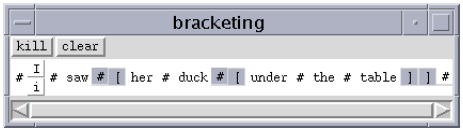

Figure 8. Bracket Window for *I saw her duck under the table*.

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 delimiters. Clicking on a pair of these buttons inserts a pair of brackets that encloses the material in between them. This imposes a filter on the trees to be 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 Figure 8 in which *her duck under the table* and *under the table* must form consituents. There is only one valid tree corresponding to these requirements: the one in which *under the table* is an ADJUNCT of *duck*. 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 Figure 9 in which the part of speech of *her* is specified to be a personal pronoun, by requiring all the categories PRON and NP to be in, and not a possessive pronoun, by requiring the category PRONposs to be out.

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

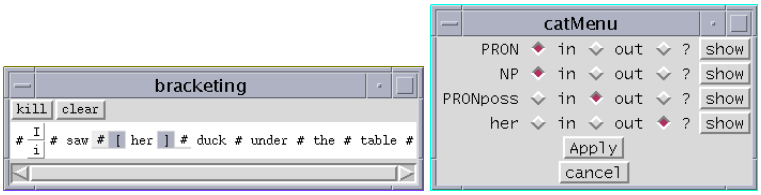

Figure 9. Bracket Window with category constraints for *her*.

constraints is also often the first step before applying 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., personal vs. possessive pronoun), 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 constituency. 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

Searching through c-structure trees, with or without the aid of the bracketing window, is ideal for certain applications. However, 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 3.2, 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.

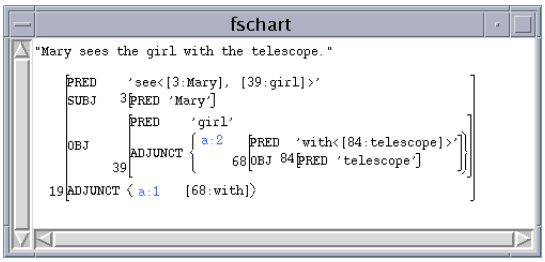

Figure 10. F-structure chart for *Mary sees the girl with the telescope*.

4.3.1. *Packed F-Structure*

The f-structure chart window (the top right window in Figure 1) 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 Figure 2 above, repeated as Figure 10. (Note that Figure 10 shows only the PRED values; this display option is discussed below.) 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*).

In Figure 11, a part of the f-structure chart for *I saw her duck under the table* is seen. The *a:3* vs. *a:1/a:2* choice reflects the separate argument frames of *see*. The *a:1/a:2* choice has a number of correlated effects on the structure: in *a:1* and *a:2*, *her duck* functions as the object. In *a:2*, the PP is adjoined to the verb, hence the f-structure with PRED-value 'under' is in the ADJUNCT set of the verb *see*; in choices *a:1* and *a:3*, this f-structure appears as an adjunct to the noun *duck* and the verb *duck*, respectively (this is not seen in the part of the chart in the window). The choice labels are color-coded: one consistent selection of choices is highlighted in red (e.g., *a:1*), all other choice labels are blue.[4] 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 "PRED only" menu item suppresses all of the attributes except PRED, the governable attributes (i.e., grammatical relations), and the semantic attributes (e.g., adjuncts). This display is 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 useful with large f-structures and multiple dependent context choices since it gives the

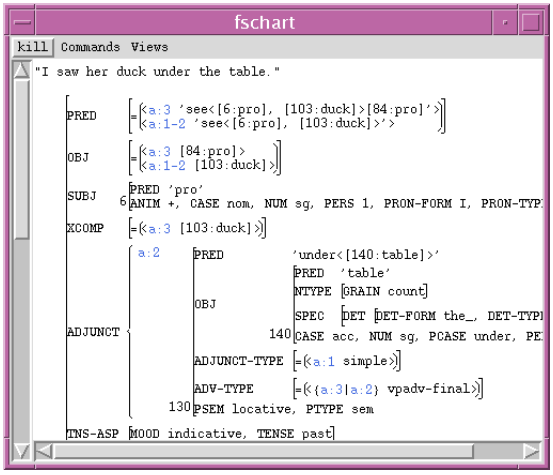

Figure 11. F-structure Chart Window for *I saw her duck under the table*.

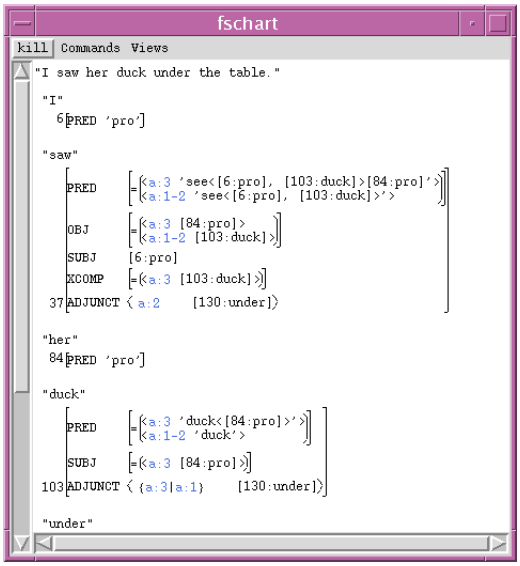

Figure 12. Linear Display for *I saw her duck under the table*.

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 saw her duck under the table* is shown in Figure 12.

4.3.2. *Choice Window*

An alternative view of the choices shown in the packed f-structure window is displayed in the f-structure chart choices window (the bottom right window in the overall layout, cf. Figure 1). The choices window indexes the packed solutions by

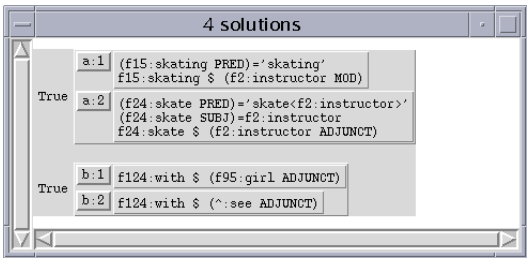

Figure 13. Choice Window for *The skating instructor saw the girl with the telescope*.

the alternative choices and displays them as a logical decision 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 Figure 13 it is easy to see that the four readings for the sentence *The skating instructor saw the girl with the telescope* arise from two independent sources: Each of the two choices *a:1* (with an instructor of skating) and *a:2* (with an instructor who is skating) can be combined with either of the PP attachment possibilities. With highly ambiguous sentences, such dependencies are very instructive, not only from the point of view of linguistic modelling but also for detecting sources of efficiency problems in parsing.

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 there is a spurious ambiguity in the grammar with respect to the given sentence.

4.4. DETECTING AMBIGUITY SOURCES

The tools described so far 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 pinpoint 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 command (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 provide 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 pass the subtree identifier to print-ambiguity-sources to find out what ambiguities are being multiplied together to produce the solutions.

(8) shows the result of print-ambiguity-sources for the sentence *Mary sees the girl with the telescope*. For example, the first line indicates that N_BASE (the noun stem for *girl*) is two way ambiguous and points to the location of the lexical entry for this word stem (line 4988 of the file eng-ncount-lex.lfg). The "perhaps" in (8) is because of the possibility that the ambiguity is the result of functional uncertainty, which cannot be detected by the tool; in fact, the ambiguity in this example is not a result of the lexical entry for the noun, but the (non)attachment of the PP to the head noun. More helpful is the second line which states that VPv[fin] is ambiguous; this information allows the grammar writer to focus on the possible instantiations of this node, concentrating on subtrees 4 and 24.

(8)
```
print-ambiguity-sources
N_BASE:57:1 adds a 2-way ambiguity, perhaps from line 4988 in
eng-ncount-lex.lfg
VPv[fin]:1040 is ambiguous because of subtrees 4 & 24
```

5. Comparison to related approaches

XLE's selection and sorting facilities described in section 4.3 are very similar in spirit to the display and selection displays of the TreeBanker tool within the Core Language Engine (CLE), described in Carter (1997). There are however certain differences between the two approaches which we discuss here.

Since multiple analyses are generally grounded in a small number of independent variations, the TreeBanker only presents ambiguity-inducing discriminating properties to the user, who then selects appropriate properties to filter out incorrect analyses, and finally selects the "good" analysis from the set of alternatives.

A characteristic feature of the TreeBanker's approach is that the properties used for selection are not only properties in CLE's underlying QLF (Quasi-Logical Form) representation. The properties can be smaller pieces of information, extracted from the QLF or the underlying syntax tree, and can characterize different types of linguistic information: constituency, predicate-argument relations, word senses, sentence types, and grammar rules used. It is the set of these more abstract extracted properties, not the QLF's themselves, which are stored in the database of analyzed sentences and presented to the user for disambiguation. In this way, the TreeBanker disambiguation tool is independent of the specific underlying representation format, and is easily adapted to different kinds of representation formats. This modular design of the TreeBanker tool differs from the more integrated XLE approach.

A distinctive feature of the TreeBanker's modular set-up is that an independent reasoning process is responsible for filtering the "good" analysis from the pool

of proposed analyses, on the basis of the properties selected by the user, using propagation rules of the form: if property *p* is marked as bad, all analyses with property *p* are marked as bad; if a property *p* is selected as good, analyses that do not have *p* are filtered out as bad (on the assumption that there is only a single good analysis), etc. A weakness of this independent reasoning process is that the set of propagation rules is based on the assumption of a single "good" analysis and cannot reliably handle truly ambiguous sentences. By contrast, XLE's integrated approach applies sound techniques for ambiguity packing, both in the processing and representation of ambiguities, and correctly handles truly ambiguous sentences.

The fact that the TreeBanker prompts the user with higher-level, abstract linguistic properties for selection is a very attractive feature. It makes the tool convenient for non-expert, but linguistically trained users. In the XLE environment the chart displays present choices to the user which consist of the very concrete LFG syntactic analysis features defined in the grammar. It is straightfoward for the grammar expert to select from these analyses. For treebanking tasks, however, non-expert users have to become familiar with relevant grammar specific properties in order to select solutions from the chart.

As an attempt to combine XLE's strengths with the attractive features offered by the TreeBanker tool, the current XLE set-up could be tailored towards a more modular interface structure between grammatical analysis and display for non-expert user selection. Similar to the Optimality projection used in Frank et al. (2001) for OT-based filtering of competing analyses, a new *properties* projection *p* could be used to associate syntactically complex properties of the c- and/or f-structure with more abstract notions, to be presented to the user in a separate chart window as selection criteria.

For example, the VP and NP rules could be augmented to produce an additional representation providing predicates like *argument_of* and *modifier_of*.[5] Then the PP attachment ambiguity of the example *Mary sees the girl with the telescope* of Figures 2 and 3 would trigger contexted properties a non-expert user can easily choose from.

$$
\begin{array}{lll}
 & \text{TRUE} & \rightarrow \text{argument\_of(see,girl)} \\
\wedge & a{:}1 & \rightarrow \text{modifier\_of(girl,with,telescope)} \\
\wedge & a{:}2 & \rightarrow \text{modifier\_of(see,with,telescope)}
\end{array}
$$

6. 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 efficiently 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 both tasks, the tools were used with success by people with some linguistic knowledge but who were unfamiliar with LFG or with grammar writing in general.

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.

Acknowledgements

We would like to thank the ESSLLI 2000 audience and the reviewers for their detailed comments.

A. Frank was at XRCE in Grenoble, and J. Kuhn was at IMS Stuttgart, when the original version of this paper was written.

Notes

[1] See also the survey of grammar workbenches given in Volk et al. (1994).

[2] 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.

[3] Context variables for disjunctions are systematically named by letters *a, b, c* with numbers for each local disjunct. *a:1* to *a:n* are internally represented as alternative disjunctive contexts. Thus, if *a:1* and *a:2* are the only *a*-context variables in a disjunction, $(a{:}1 \rightarrow fact_1) \wedge (a{:}2 \rightarrow fact_2)$ is internally compiled to: $(a \rightarrow fact_1) \wedge (\neg a \rightarrow fact_2)$. Disjunctions with more than two options are handled by introducing further internal context variables (e.g. $a{:}1 = p$, $a{:}2 = \neg p \wedge q$, $a{:}3 = \neg p \wedge \neg q$).

[4] 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.

[5] In the XLE specification format, the rules would be augmented with special clauses to represent abstract properties in the *p* projection, as sketched below (predicate names in semantic forms (i.e. PRED features) are referred to by PRED FN (functor name)):

```
VP → V    ↑ = ↓
       NP: (↑ OBJ) = ↓
           argument_of((↑ PRED FN),(↓ PRED FN)) ∈ p::*;
       ...
       PP: { (↑ OBL) = ↓
             argument_of((↑ PRED FN), (↓ PRED FN), (↓ OBJ PRED FN)) ∈ p::*
           | ↓ ∈ (↑ ADJUNCT)
             modifier_of((↑ PRED FN),(↓ PRED FN),(↓ OBJ PRED FN)) ∈ p::* }
       ...

NP → N    ↑ = ↓
       PP: ↓ ∈ (↑ ADJUNCT)
           modifier_of((↑ PRED FN),(↓ PRED FN),(↓ OBJ PRED FN)) ∈ p::*
       ...
```

References

Alshawi H. (ed.) (1992) *The Core Language Engine*. MIT Press, Cambridge, MA.

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

Bresnan J. (2000) *Lexical-Functional Syntax*. Blackwell Publishers.

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

Carroll J., Briscoe E., Grover C. (1991) *A Development Environment for Large Natural Language Grammars*. Computer Laboratory, Cambridge University, UK, Technical Report 233.

Carpenter B., Penn G. (1999) *ALE: The Attribute Logic Engine User's Guide*. Bell Laboratories, NJ.

Carter D. (1997) The TreeBanker: A Tool for Supervised Training of Parsed Corpora. In: *Proceedings of the ACL Workshop on Computational Environments for Grammar Development and Linguistic Engineering*, Madrid, Spain.

Clement L. (1996–1999) XLFG – mode d'emploi. Ms. Université Paris 7. http://talana.linguist.jussieu.fr/~lionel/xlfg/xlfg.html.

Copestake A., Carroll J., Malouf R., Oepen S. et al. (1999/2000) The (New) LKB System. Ms. CSLI. http://www-csli.stanford.edu/~aac/doc5-2a.pdf.

Copestake A., Flickinger D. (2000) An Open-Source Grammar Development Environment and Broad-Coverage English Grammar Using HPSG. In: *Proceedings of the Second Conference on Language Resources and Evaluation (LREC-2000)*, Athens, Greece.

Dipper S. (2000) Grammar-Based Corpus Annotation. In *Proceedings of the Workshop on Linguistically Interpreted Corpora*, Luxembourg.

Doran C., Egedi D., Hockey B. A., Srinivas B., Zaidel M. (1994) XTAG System – a Wide Coverage Grammar for English. In: *Proceedings of the 15th COLING*, Kyoto, Japan.

Dörre J., Dorna M., Junger J., Schneider K. (1996) *The CUF User's Manual*. Institut für maschinelle Sprachverarbeitung, University of Stuttgart, Germany.

Emele M. (1994) The Typed Feature Structure Representation Formalism. In *Proceedings of the International Workshop on Sharable Natural Language Resources*, Ikoma, Nara, Japa.

Flickinger D. (2000). On Building a More Efficient Grammar by Exploiting Types. In Boguraev B., Garibliano R., Tait J. (eds.), *Special Issue on Efficient Processing with HPSG: Methods, Systems, Evaluation, Natural Language Engineering*, Vol. 6(1), Cambridge University Press, pp. 15–28.

Frank A., King T. H., Kuhn J., Maxwell J. (2001) Optimality Theory Style Constraint Ranking in Large-Scale LFG Grammars. In Sells P. (ed.), *Formal and Empirical Issues in Optimality Theoretic Syntax*, CSLI Publications, Stanford, CA, pp. 367–397.

Götz T., Meurers D., Gerdemann D. (1997) *Documentation of the ConTroll System*. The ConTroll Manual. University of Tübingen, Germany.

Kuhn J. (1998) Towards Data-Intensive Testing of a Broad-Coverage LFG Grammar. In Schröder B., Lenders W., Hess W., Portele T. (eds.), *Computers, Linguistics, and Phonetics between Language and Speech, Proceedings of the 4th Conference on Natural Language Processing – KONVENS-98*, Peter Lang, Bonn, pp. 43–56.

Lehmann S., Oepen S., Regnier-Prost S., Netter K., Lux V., Klein J., Falkedal K., Fouvry F., Estival D., Dauphin E., Compagnion H., Baur J., Balkan L., Arnold D. (1996) TSNLP – Test Suites for Natural Language Processing. In *Proceedings of COLING 1996*, Kopenhagen, pp. 711–716.

Maxwell J., Kaplan, R. (1989) An Overview of Disjunctive Constraint Satisfaction. In *Proceedings of the International Workshop on Parsing Technologies*. Also published as: A method for disjunctive constraint satisfaction. In Tomita M. (ed.), *Current Issues in Parsing Technology*, Kluwer Academic Publishers, 1991.

Maxwell J., Kaplan R. (1993) The Interface between Phrasal and Functional Constraints. *Computational Linguistics*, 19/4, pp. 571–590.

Maxwell J., Kaplan R. (1996) Unification-Based Parsers that Automatically Take Advantage of Context Freeness. Paper Presented at the *LFG96 Conference*, Grenoble, France. Ms. Xerox PARC.

Oepen S., Callmeier U. (2000) Measure for Measure: Parser Cross-Fertilization. Towards Increased Component Comparability and Exchange. In *Proceedings of the 6th International Parsing Technology Workshop*, Trento, Italy, pp. 183–194.

Oepen S., Carroll, J (2000) Ambiguity Packing in Constraint-based Parsing – Practical Results. In *Proceedings of NAACL*, Seattle, WA, pp. 162–169.

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

Volk M., Jung M., Richarz D., Fitschen A., Hubrich J., Lieske C., Pieper S., Ridder H., Wagner, A. (1994) *GTU – A Workbench for the Development of Natural Language Grammars*. Institute of Computational Linguistics: University of Koblenz-Landau.

XTAG Research Group (1995/1998/2001) *A Lexicalized Tree Adjoining Grammar for English*. IRCS, University of Pennsylvania. IRCS-95-03/IRCS-98-18/IRCS-01-03, http://www.ircs.upenn.edu/reports/.