DESIGNING AND IMPLEMENTING DISCRIMINANTS FOR LFG GRAMMARS

Victoria Rosén, Paul Meurer and Koenraad de Smedt
University of Bergen and Unifob AKSIS

Proceedings of the LFG07 Conference

Miriam Butt and Tracy Holloway King (Editors)

2007

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

We extend discriminant-based disambiguation techniques to LFG grammars. We present the design and implementation of lexical, morphological, c-structure and f-structure discriminants for an LFG-based parser. Chief considerations in the computation of discriminants are capturing all distinctions between analyses and relating linguistic properties to words in the string. Our work is mostly tested on Norwegian, but our approach is independent of the language and grammar.

1 Introduction

The use of linguistically motivated handwritten grammars in realistic applications is dependent on the capacity to automatically resolve ambiguities produced by the grammar. Statistical techniques for disambiguation by parse ranking require training of the parser on a previously analyzed and disambiguated corpus—a treebank. Quality controlled treebanks that can serve as gold standards cannot be constructed without considerable manual effort towards ambiguity resolution. Intelligent ways of minimizing these efforts have been the subject of earlier research in the context of different tasks and formalisms (Carter, 1997; Van der Beek et al., 2002; Oepen et al., 2004). In our work on treebanking by automatically parsing a corpus with an LFG grammar, we have employed and further developed such techniques.

In this paper we explain in depth how discriminants can be extended to LFG grammars and how we have implemented them. The paper is structured as follows. First we present previous work on discriminants. Then we describe our design of various types of discriminants for LFG grammars. These will be illustrated and motivated with examples parsed with the Norwegian grammar developed at the University of Bergen within the Parallel Grammar project (Butt et al., 2002). Furthermore, we describe their implementation, i.e. the computation of discriminants from linguistic structures. Finally, we discuss the presentation and use of discriminants. The LFG Parsebanker, a toolkit developed at the University of Bergen in the TREPIL[1] and LOGON[2] projects, implements the computation and presentation of LFG discriminants.

2 Previous Work on Discriminants

Discriminant-based disambiguation was first presented by Carter (1997) as a time-saving method for treebanking. Carter's aim was to train a linguistic analyzer for several domains and tasks, each one requiring a separate analyzed and disambiguated corpus. In this context, it is clearly desirable to optimize the efficiency

†This work was supported in part by a grant from the Research Council of Norway. We would like to thank John Maxwell at PARC for his help with implementation issues.

[1]http://gandalf.aksis.uib.no/trepil

[2]http://www.emmtee.net/

of manual disambiguation. Inspecting full analyses proved to be "a tedious and time-consuming task". In contrast, a few lexical or structural properties are often sufficient to distinguish the one intended analysis from many other analyses. Examples of properties that involve relatively simple choices are PP attachment, word senses, and the arity of predicates. Calling such distinguishing properties *discriminants*, Carter implemented their identification and presentation in his TreeBanker tool. He designed various discriminants, including constituents, semantic triples, word senses, sentence types, and grammar rules used.

In the TreeBanker's graphical interface, the user can label discriminants as either good or bad, or can leave them undecided. Carter defined a set of inferencing rules based on these decisions. If a discriminant is marked by the user as bad, then all analyses that contain this property are rejected, whereas if a discriminant is marked by the user as good, then only analyses that contain it are kept. Thus, the set of analyses is narrowed down, until only one analysis remains. Furthermore, a discriminant that is true only of analyses that have already been rejected must be bad. Conversely, a discriminant that is true of all the still undecided analyses must be good (assuming there is at least one good analysis). In the cases where a discriminant is inferred to be either good or bad for all analyses, it loses its discriminatory power, i.e. it is trivial, and hence it need not be presented to the user, who can thus concentrate on more relevant choices.

Carter pays special attention to "user-friendly" discriminants which are easy for humans to judge and are prominent in the display. The efficiency of this method, as compared to presenting all the full analyses to the user, can be appreciated from the fact that a combination of a small number of local ambiguities can result in a large number of analyses. Carter mentions an example with 154 analyses, for which 318 discriminants are computed, yet only two discriminant choices are necessary to select the correct analysis.

Discriminants have also been used in at least two other projects, both HPSG-based. In the context of the Alpino project (Van der Beek et al., 2002), a large treebank was built using manual disambiguation based on Carter's principles but with a different design. Lexical discriminants, representing ambiguities that result from lexical analysis, are always presented to the annotator first, because it is claimed that lexical decisions are easy to make. Furthermore, constituent discriminants represent alternative groupings of words in constituents, and dependency triples represent alternative paths in a dependency tree. These can be compared to our c-structure and f-structure discriminants, which will be presented in sections 3.2 and 3.3 respectively.

The LinGO Redwoods project (Oepen et al., 2004) was aimed at building a dynamic treebank as a testbed for grammar development. Since grammar development presupposes frequent automatic reparsing of a corpus, automatic redisambiguation is highly desirable. This was achieved by storing the annotator's discriminant choices and reapplying them when reparsing. To our knowledge, LinGO Redwoods was the first project to closely integrate treebanking and grammar development in this way. Properties related to constituents (i.e. use of a grammar rule

over a specific substring), lexical items (part of speech), semantics (primary predicate) and node labeling were used as discriminants. With the help of a suitable tool for identifying and presenting these discriminants, an annotator performance of about 2000 sentences per week was achieved.

In all work with discriminants, Carter's rules for narrowing down the set of analyses based on the annotator's choice of discriminants, as well as his rules for narrowing down the set of discriminants so that only the nontrivial ones are kept, are essential. But even though, by means of Carter's rules, enough discriminant choices will eventually lead to a single analysis, this analysis is not necessarily the correct one. There may be no correct analysis among the ones that the parser produced, or a wrong discriminant choice could have eliminated the correct analysis. To assist the annotator in making the right choices, a sophisticated, user-friendly tool that identifies and presents discriminants together with specific analyses is indispensable. Both the TreeBanker and the discriminant tools used in Alpino and LinGO Redwoods aim to provide such assistance in the context of the grammars and parsers they operate with. However, the types of discriminants, their computation, and even their presentation are not universal, but depend on the grammar formalism, the parser, and on user-oriented and system-oriented design choices. To our knowledge, there has been no previous work on designing discriminants for LFG grammars and implementing them for an LFG-based parser such as the Xerox Linguistic Environment (XLE) (Maxwell and Kaplan, 1993).

3 Designing Discriminants for LFG

The number of analyses of realistic sentences provided by a grammar may run into the thousands. In such cases, disambiguation by the sequential inspection of individual structures is prohibitively time consuming. XLE provides packed c- and f-structures which are compact representations of all the information in all analyses. In XLE's native interface it is possible to disambiguate interactively by choosing between alternatives indicated in the packed structures. While an important property of packed structures is that they are concise from a computing standpoint (Maxwell and Kaplan, 1993), this property is nevertheless of little help towards efficient manual disambiguation, since for sentences with multiple ambiguities, packed structures may become too unwieldy for a human to cope with. Disambiguation with discriminants does not suffer from the complexity issue that packed structures have, since each discriminant is local and may be chosen independently of all others.

There are often a large number of elementary properties that are not shared by all analyses, such as local c-structure node configurations and labels or f-structure attributes and values. Any such elementary property is a candidate for being a discriminant, for all such properties actually discriminate between analyses. However, in many cases it is impossible for a human disambiguator to pick out such elementary properties in isolation. In order for them to be reliably recognizable as proper-

ties of the intended analysis, they must be related to words in the string. This is a crucial point in the design of discriminants.

The present work on discriminants is focused on how they may be defined and used in an optimal way for LFG grammars. Discriminants should be designed so as to automatically identify all possible distinctions between analyses and make these recognizable to the annotator. It is important that the discriminants contain enough information to make it possible to uniquely identify them, but little enough information that they remain elementary local properties. The graphs representing the c-structure and f-structure must be fully traversed to find all possible distinctions between structures. We have defined four major types of discriminants for LFG grammars: lexical discriminants, morphological discriminants, c-structure discriminants and f-structure discriminants.

3.1 Lexical and Morphological Discriminants

We agree with Van der Beek et al. (2002) that lexical ambiguities are often the easiest to resolve. Two types of discriminants are meant to aid in resolving lexical ambiguities: lexical discriminants and morphological discriminants.

A *lexical discriminant* is a word form with its lexical category. Consider the Norwegian sentence in example (1) and its two c-structures in figure 1.

(1) *Glade fisker svømmer.*
Glad fish swim/swimmer
"Glad fish swim." / "Glad ones fish a swimmer."

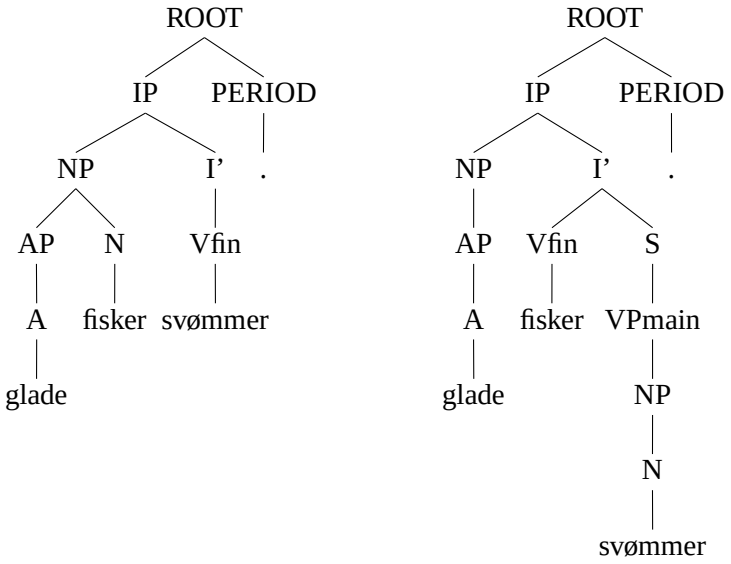

Figure 1: Two analyses for example (1), the left one corresponding to 'Glad fish swim', the right one to 'Glad ones fish a swimmer'

In this example, both *fisker* and *svømmer* may be either a noun or a verb, and because of this there are two quite different c-structures. The entire c-structures need not be examined, however, since determining the lexical category of either of these words is enough to determine which c-structure is the intended one. The relevant subtrees containing preterminal and terminal nodes for example (1) are shown in figure 2. Table 1 illustrates the representation of lexical discriminants for this example.

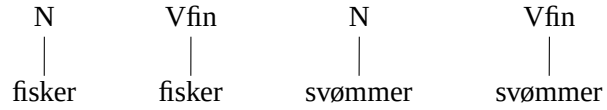

Figure 2: Subtrees defining lexical discriminants for example (1)

Table 1: Representation of lexical discriminants for example (1)

'fisker': N
'fisker': Vfin
'svømmer': N
'svømmer': Vfin

The lexical category specified in the discriminant is sometimes simply the traditional part of speech (e.g. N), sometimes a more fine-grained category (e.g. Vfin). Whatever preterminal node label occurs in the subtree will be the category in the discriminant.

Sometimes a word form may be ambiguous between different lexemes or between different forms of one lexeme within the same part of speech. This is the case in the present example. Even after the category N has been chosen by selecting the first discriminant in table 1, the word form *fisker* may still be an inflected form of the noun *fisk* "fish" or of the noun *fiske* "fishing". Since lexical discriminants are not sufficient for the disambiguation of lexical ambiguities, we also define morphological discriminants. A *morphological discriminant* is a word with the tags it receives from morphological preprocessing. The two morphological analyses for the noun *fisker* are illustrated in figure 3, which shows a simplified version of the sublexical trees not usually displayed by XLE. The morphological discriminants for this example are represented as in table 2.

Table 2: Morphological discriminants for *fisker* in example (1)

fisk+Noun+Masc+Indef+Pl
fiske+Noun+Neut+Indef+Pl

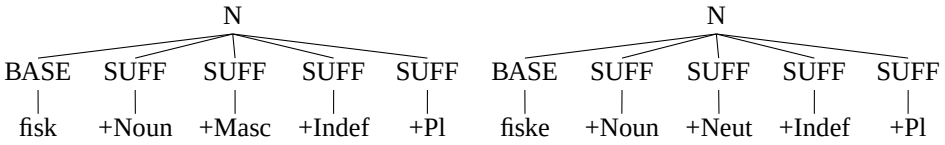

Figure 3: Two morphological analyses for *fisker*

Neither lexical nor morphological discriminants alone are sufficient for the full disambiguation of lexical ambiguities. As shown in the above examples, lexical discriminants cannot distinguish between different word forms that have different features and/or base forms but the same lexical category. In this example, full lexical disambiguation could have been achieved through selecting only morphological discriminants. This is not always the case, however, since not all words go through morphological preprocessing. For some words, morphosyntactic features may be directly encoded in the lexical entry. Therefore both lexical and morphological discriminants are necessary for lexical disambiguation. There are also cases where lexical ambiguities remain after all lexical and morphological discriminants have been chosen; we will return to these in section 3.3.

3.2 C-structure Discriminants

C-structure discriminants are important for the disambiguation of syntactic ambiguities. Their design aims at selecting an elementary local property of a tree. They are therefore based on minimal subtrees, a minimal subtree being defined as a mother node and her daughters. Since identical subtrees may occur more than once in the same analysis, these need to be related to the substring that they dominate. Example (2) involves two different PP attachment choices, as shown in the c-structure trees in figure 4. The substring which is relevant for the disambiguation of this example is shown with its bracketing in example (3). The simple breakdown of the substring into its immediate constituents is shown in the unlabeled bracketing in (3a), and the two different PP attachments are shown in the labeled bracketings in (3b) and (3c).

(2) *Vi fanget fisk med stang.*
We caught fish with fishing-rod
"We caught fish with a fishing rod."

(3) (a) [[fisk] [med stang]]
(b) [$_{VPmain}$ [$_{NP}$ fisk] [$_{PP}$ med stang]]
(c) [$_{NP}$ [$_{N}$ fisk] [$_{PP}$ med stang]]

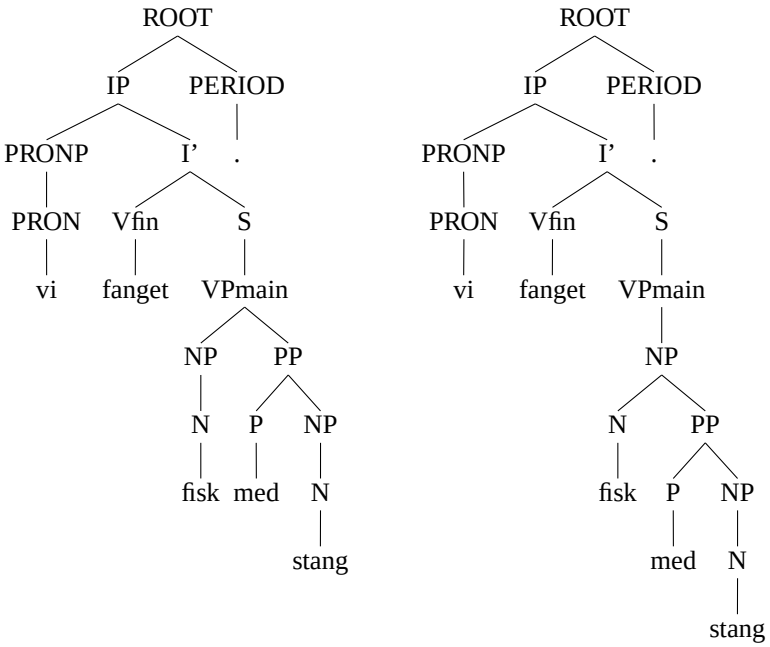

Figure 4: Two PP attachments for example (2)

C-structure discriminants are of two subtypes. An unlabeled top-level bracketing of a constituent substring is a *constituent discriminant*. A top-level bracketing of a constituent substring labeled by the rule which induces that bracketing is a *rule discriminant*. The c-structure discriminants for this example are shown in table 3.

Table 3: C-structure discriminants for the PP attachments in figure 4

fisk || med stang
VPmain → NP PP
NP → N PP

The top row in this table shows the representation of the constituent discriminant corresponding to the bracketed string in (3a). Instead of indicating the bracketing by enclosing the constituents in square brackets, the constituents are separated by two vertical bars. The second and third rows of the table illustrate the representation of rule discriminants, with the second row corresponding to (3b) and the third row corresponding to (3c). The representation of rule discriminants is simply expressed as a grammar rule, but this rule must be interpreted as the labeled bracketing of the string in question. Since rule discriminants are always displayed in a

table cell underneath the corresponding constituent discriminant, it is always clear which substring the rule applies to.

Both types of c-structure discriminants can be useful: sometimes it is possible for an annotator to decide on the labeling as well as the bracketing, while in other cases one may wish to commit to a bracketing but not to a certain labeling. In the case in table 3, however, the constituent discriminant is actually trivial. Since both analyses share this constituent structure, the bracketing [[fisk] [med stang]] does not discriminate between analyses.

3.3 F-structure Discriminants

A c-structure may project more than one f-structure. In example (4), the constituent *hver time* may function as either OBJ or ADJUNCT.

(4) *Vi spiser hver time.*
we eat every hour
"We eat every hour."

F-structure discriminants are based on partial paths through f-structures. For f-structures it is not so apparent as for c-structures how to make local properties easily identifiable in discriminants, since the string is not represented in the f-structure. Therefore, the design of f-structure discriminants crucially exploits PRED values, which typically provide the most direct connection to words in the string. An f-structure discriminant is a minimal path through the f-structure from a PRED value to another PRED value or to an atomic value, a minimal path being one that does not cross any intermediate PRED values and does not contain cycles.

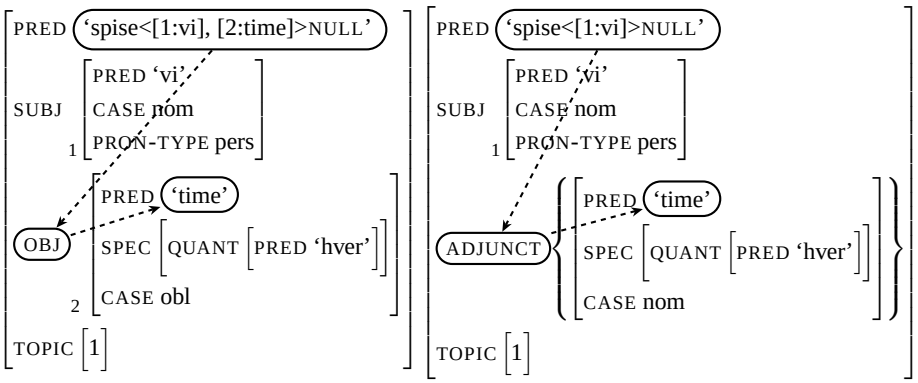

Figure 5: Simplified f-structures for example (4)

Table 4 represents some relevant f-structure discriminants for the example in figure 5. The empty brackets in the PRED values show the arity of the predicate. The first discriminant may thus be read: *the two-place predicate 'spise' has an*

object whose PRED *value is 'time'*, while the second discriminant may be read: *the one-place predicate 'spise' has a set of adjuncts, one of which has the* PRED *value 'time'*. The PRED attributes themselves are omitted in the discriminants for brevity.

Table 4: Some f-structure discriminants for example (4)

'spise<[],[]>NULL' OBJ 'time'
'spise<[]>NULL' ADJUNCT > 'time'

The path in an f-structure discriminant is, however, not always from PRED value to PRED value. The word *barn* in example (5) is ambiguous between singular and plural, and the morphology tells us that not by assigning different morphological subtrees but by assigning a single tag *SP* representing both singular and plural. For this single tag, the rules in the grammar assign two different values, as shown in the packed f-structure in figure 6, where parentheses surround the alternate values for the number attribute. Since there are neither lexical nor morphological discriminants in cases like this, we must let f-structure discriminants describe paths from PRED values to atomic values, as shown in table 5.

(5) *Vi liker barn.*
We like child-SG/PL
"We like child/children."

$$\begin{bmatrix} \text{PRED} & \text{'like<[3:vi], [5:barn]>NULL'} \\ \text{SUBJ} & 3\begin{bmatrix} \text{PRED} & \text{'vi'} \\ \text{CASE} & \text{nom} \\ \text{PRON-TYPE} & \text{pers} \end{bmatrix} \\ \text{OBJ} & 5\begin{bmatrix} \text{PRED} & \text{'barn'} \\ \text{NUM} & \begin{pmatrix} a_1 & \text{pl} \\ a_2 & \text{sg} \end{pmatrix} \end{bmatrix} \\ \text{TOPIC} & \boxed{3} \end{bmatrix}$$

Figure 6: Simplified packed f-structure for example (5)

Table 5: F-structure discriminants with atomic values for *barn*

'barn' NUM sg
'barn' NUM pl

Moreover, as mentioned earlier, not all words go through morphological preprocessing. Some words receive multiple features directly through a disjunction in the lexicon. An example is *den*, which can either be a demonstrative meaning "that" or an article meaning "the". A simplified partial lexical entry for this word is shown in example (6).

(6) den D {(↑SPEC DET DET-TYPE) = demon
| (↑SPEC DET DET-TYPE) = article}

Since both have the category D (determiner) there are no lexical discriminants, and since this word does not go through morphological preprocessing, there are no morphological discriminants either. This ambiguity can therefore only be resolved in the f-structure. Two of the f-structure discriminants for *den* are shown in table 6.

Table 6: F-structure discriminants with atomic values for *den*

'den' DET-TYPE demon
'den' DET-TYPE article

The previous two cases have shown the necessity of allowing f-structure discriminants based on a minimal path from a PRED value to an atomic value. Since in general we do not know what atomic values will provide the only means of resolving an ambiguity for any grammar and any language, we have to allow every path from a PRED value to an atomic value to be a discriminant candidate. This gives rise to a very large number of discriminants with a high degree of redundancy. Nevertheless, the disadvantage of the large number of discriminants is outweighed by the assurance of having discriminants for all possible distinctions.[3] Furthermore, the number of redundant discriminants quickly diminishes as discriminant choices are made.

3.4 Discriminant Anchors

Each type of discriminant is designed so that it relates linguistic properties to words in the string in order to make it easy to recognize the desired properties. However,

[3]There are marginal cases where two differing c-structures or f-structures will have no discriminants, but these cases are very unlikely to occur with a real grammar. In concrete terms, the two (sub-)c-structures A → B → A → X and A → B → A → B → A → X are different but cannot be distinguished by discriminants; it is easy to see that all other cases are extensions of this example. A simple example of two differing f-structures that cannot be distinguished by (our) discriminants is given by the following pair: $\begin{bmatrix} \text{A } {}_1x \\ \text{B } \begin{bmatrix} 1 \end{bmatrix} \end{bmatrix}$ and $\begin{bmatrix} \text{A } x \\ \text{B } x \end{bmatrix}$. All other non-cyclic examples have in common with the given minimal one that the tree expansions of both f-structures are identical, that is, the f-structures only differ in whether two attributes share their values or have (distinct) values with identical expansions. The situation with f-structures containing cycles is somewhat more complicated, but comparable.

the same word or substring may occur more than once in the same string. In order to allow the correct identification, and hence, disambiguation, of identical substrings, discriminants are *anchored* to their string positions in terms of character count (which for technical reasons is the least problematic to calculate).

Consider the repeated word *fisker* in example (7). If we did not take string position into account, these two occurrences of the same word form would result in identical discriminants. By anchoring the discriminants in string positions as illustrated in table 7, identical substrings can always be disambiguated correctly. The anchors *10* and *31* refer to the position of the first character in the word *fisker* in its two occurrences.

(7) *De* *store* *fisker* *spiser* *de*
you/they/the/that big fish.N/fishing.N/fish.V eater.N/eat.V the/that
små *fisker.*
small fish.N/fishing.N/fish.V
"The big fish eat the small fish."/ "The small fish, the big fish eat."/ "Those big fish eat the small fish."/etc.

Table 7: Anchored morphology discriminants for the word *fisker* in example (7)

10	'fisker': N
10	'fisker': Vfin
31	'fisker': N
31	'fisker': Vfin

In some cases, a single anchor is not sufficient to ensure that discriminants that should be distinct actually are distinct. Consider again example (7), and assume that the noun discriminants have been chosen for both occurrences of *fisker*. Since Norwegian is a V2 language, we are still left with an ambiguity as to which NP is the SUBJ and which is the OBJ. The f-structure discriminants shown in table 8 have two anchors. The first anchor refers to the position of the verb *spiser* which projects the PRED value 'spise<[],[]>NULL'. The second anchor refers to the position of the noun *fisker* which projects the PRED value 'fisk'. Doubly anchored discriminants are those which are paths from PRED value to PRED value.

Table 8: Doubly anchored f-structure discriminants for example (7)

17:10	'spise<[],[]>NULL' SUBJ 'fisk'
17:31	'spise<[],[]>NULL' SUBJ 'fisk'
17:10	'spise<[],[]>NULL' OBJ 'fisk'
17:31	'spise<[],[]>NULL' OBJ 'fisk'

4 Calculation of Discriminants

Discriminants are calculated on the basis of packed c- and f-structures, which are internally represented as directed (not necessarily acyclic) graphs, where each node is labeled with the context (the set of solutions) for which it is valid. This means that the c- and f-structures for a given solution may be recovered by discarding all nodes whose context does not contain that solution. It is, however, crucial to note that neither the discriminants themselves nor the algorithm that computes them depends on the solutions being packed; the algorithm uses packed solutions solely for efficiency reasons and could easily be modified to operate on unpacked c- and f-structures.

In XLE a context is represented as a set of (compatible) choices. The choices corresponding to a packed structure are organized in AND/OR graphs, and each solution corresponds to a maximal selection of compatible choices. A maximal selection can be characterized as a choice of a maximal path in each AND branch of the choice tree. Non-maximal selections correspond to sets of solutions.[4] A typical choice graph looks like figure 7, where $\wedge$ = AND and $\vee$ = OR, and a possible selection corresponding to a single solution is given by (a_2, c_1, e_1, b_2), where c_1 is redundant. The graph encodes 12 solutions.

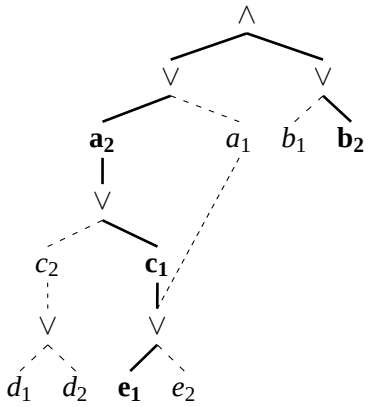

Figure 7: Choice tree with a highlighted path to a single solution

The solutions encoded in a choice graph can easily be enumerated (and thus ordered) using a depth-first multi-traversal of the graph, and a given context can thus be mapped to a bit vector that encodes solutions that are contained in the context by ones and solutions not contained in the context by zeros. For easier processing, all node contexts in the packed structures are converted to solution bit vectors.

As a first step in the calculation of discriminants, the packed graphs are traversed, and all relevant local properties are computed, each of them being associ-

[4]Note that not every solution set can be represented by a selection of compatible choices.

ated with the context in which it is valid. These local properties (together with their contexts) are called *discriminant candidates*.

Since we want to keep apart discriminant candidates with identical local properties that are related to different positions in the source string, we also record the string position from which the local property originates (the anchor of the discriminant). This is straightforward for c-structure discriminants calculated on the basis of c-structure graphs (lexical, morphological and c-structure discriminants) since c-structure lexical nodes are directly associated with string positions. In the case of f-structure discriminants, however, one first has to identify the c-structure node the semantic form of the predicate was projected from. The discriminant anchor is then given by the string position associated with the leftmost lexical node below that node.

If discriminant candidates originating from different parts of the graph have identical patterns (i.e. express the same local properties) and have the same anchors, we combine them into one new discriminant candidate whose context is the union of the original candidates' contexts. Many of the calculated discriminant candidates may be trivial, as their local properties might be valid for all solutions. Removing these trivial discriminant candidates yields the set of proper discriminants.

Let us consider a simple example. In figure 8, we see the packed c-structure and the choice tree for the four-way ambiguous string in example (8).

(8) *Det* *regnet.*
that.D/it.PRON/it.PRONexpl rain/rained/calculated
"That rain." / "It calculated." / "That (one) calculated." / "It rained."

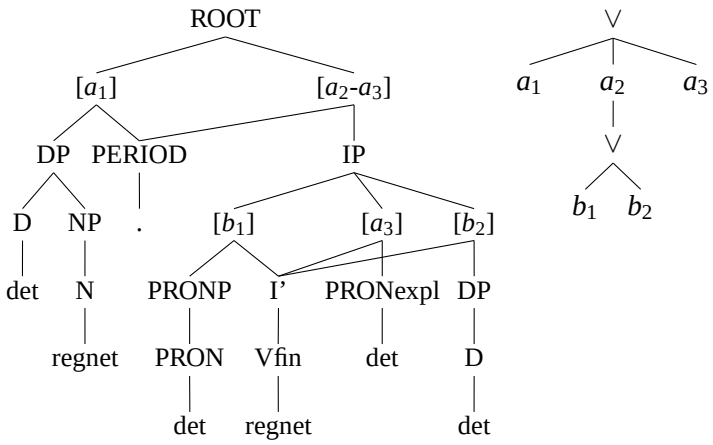

Figure 8: Packed c-structure and choice tree for example (8)

Following the algorithm outlined above, we obtain the c-structure rule and constituent discriminant candidates in table 9. All the rule discriminant candidates are

different, thus each of them is a proper discriminant. The associated constituent discriminants, however, are trivial, since the contexts of identical candidates add up to the context containing all solutions. A grouping of the resulting discriminants by identical constituents is presented in table 10. Note that despite the name *rule discriminant*, this kind of discriminant is computed exclusively on the basis of the structures, while access to the grammar rules that assigned those structures is not required.

Table 9: C-structure rule and constituent discriminant candidates for example (8)

anchor	**labeled bracketing**	**substring bracketing**	**context**	**solution vector**
1	ROOT → DP PERIOD	det regnet | .	a_1	1000
1	ROOT → IP PERIOD	det regnet | .	$a_2 - a_3$	0111
1	DP → D NP	det | regnet	a_1	1000
1	IP → PRONP I'	det | regnet	b_1	0100
1	IP → DP I'	det | regnet	b_2	0010
1	IP → PRONexpl I'	det | regnet	a_3	0001

Table 10: Grouping of c-structure discriminants for example (8)

anchor	**discriminant**	**# of solutions**	**solution vector**
1	det regnet | .	(4)	1111
1	ROOT → DP PERIOD	1	1000
1	ROOT → IP PERIOD	3	0111
1	det | regnet	(4)	1111
1	DP → D NP	1	1000
1	IP → PRONP I'	1	0100
1	IP → DP I'	1	0010
1	IP → PRONexpl I'	1	0001

The lexical and morphological discriminants are also computed from the c-structure. In table 11 the lexical discriminant candidates for example (8) are shown. Two of the discriminant candidates (those in boldface) have identical patterns and anchors, so they must be combined to give a proper discriminant.

The computation of morphological discriminants, too, is based on the packed c-structures; this time, however, the sublexical subtrees of the c-structures are considered. Each morphological feature (including the base form) of an analyzed word gives rise to a branch of a sublexical subtree. A candidate for a morphological discriminant is then the concatenation of the base form and all features that can be read off of the sublexical nodes for a given word (or, equivalently, for a given anchor position) and solution.

Table 11: Lexical discriminant candidates for example (8)

anchor	**lexical rule**	**context**	**solution vector**
1	**'det' : D**	a_1	1000
1	'det' : PRON	b_1	0100
1	**'det' : D**	b_2	0010
1	'det' : PRONexpl	a_3	0001
5	'regnet' : N	a_1	1000
5	'regnet' : Vfin	$a_2 - a_3$	0111

The word *fisker* in example (9) is ambiguous between a verb and a noun. Thus, the word *fisker* has two morphological analyses, which surface in the sublexical subtrees in figure 9. We can read off the two discriminant candidates in table 12.

(9) *Jeg fisker.*
I fisherman.N/fish.V
"I am fishing." / "I, (a) fisherman."

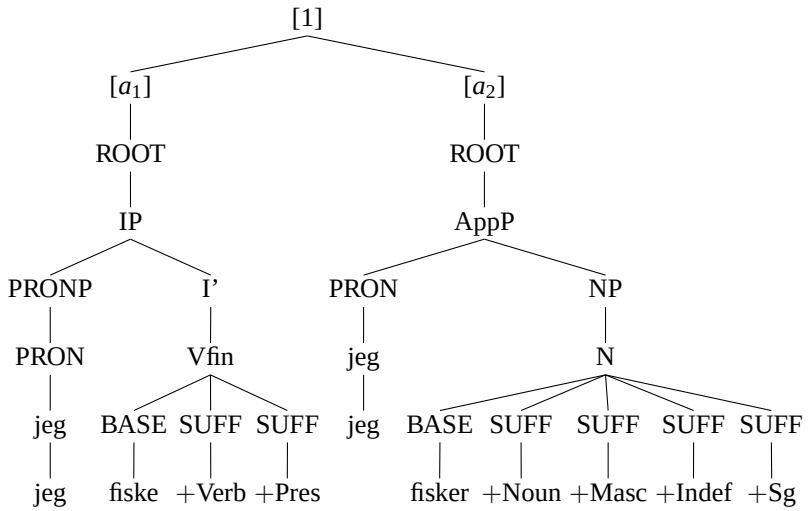

Figure 9: Packed c-structure including sublexical nodes for example (9)

It is important to bear in mind that only those words that are assigned morphological features via XLE's morphology module will have nontrivial sublexical subtrees and thus potentially give rise to morphological discriminants. Readings of ambiguous words which are directly listed in the LFG lexicon can still be disambiguated using lexical discriminants if their lexical categories are different.

Table 12: Morphological discriminant candidates for example (9)

anchor	morphology	context	solution vector
5	fiske+Verb+Pres	a_1	10
5	fisker+Noun+Masc+Indef+Sg	a_2	01

To exemplify the computation of f-structure discriminants, we consider the sentence in example (4) and its f-structures in figure 5. The relevant parts of the packed f-structure are shown in figure 10. In the packed f-structure, attribute values are annotated with the choices for which they are valid. This sentence is ambiguous, as apparent from choices a_1 and a_2, the ambiguity being manifest solely in the f-structure. An attribute in a packed structure may have more than one possible value, but the choices for those values have to be mutually exclusive, such that only one value or no value remains for each single solution. In such cases, for example the alternative PRED values indexed by a_1 and a_2 in figure 10, the set of values is enclosed in parentheses.

$$\left[\begin{array}{l}
\text{PRED} \left(\begin{array}{l} a_1 \text{ `spise<[1:vi],[2:time]>NULL'} \\ a_2 \text{ `spise<[1:vi]>NULL'} \end{array}\right) \\
\text{SUBJ}_{\,1} \left[\begin{array}{l} \text{PRED `vi'} \\ \text{PRON-TYPE pers} \end{array}\right] \\
\text{OBJ } a_1 \; {}_{2}\left[\begin{array}{l} \text{PRED `time'} \\ \text{SPEC} \left[\text{QUANT} \left[\text{PRED `hver'}\right]\right] \end{array}\right] \\
\text{ADJUNCT} \left\{ a_2 \left[2\right] \right\} \\
\text{TOPIC} \left[1\right]
\end{array}\right]$$

Figure 10: Simplified partial f-structure for example (4)

Applying the algorithm for f-structure discriminants, we obtain the candidates in table 13, which are all proper discriminants.

5 Display and Use of Discriminants

As mentioned above, a large number of discriminants may be computed for a sentence. This guarantees that there will be enough discriminants for virtually every distinction between structures, so that full disambiguation can always be achieved.

Table 13: F-structure candidates for example (4)

anchor	f-structure path	context	solution vector
0	_TOP ‘spise<[],[]>NULL’	a_1	10
0	_TOP ‘spise<[]>NULL’	a_2	01
4	‘spise<[],[]>NULL’ SUBJ ‘vi’	a_1	10
4	‘spise<[],[]>NULL’ TOPIC ‘vi’	a_1	10
4	‘spise<[],[]>NULL’ OBJ ‘time’	a_1	10
4	‘spise<[]>NULL’ SUBJ ‘vi’	a_2	01
4	‘spise<[]>NULL’ TOPIC ‘vi’	a_2	01
4	‘spise<[]>NULL’ ADJUNCT ‘time’	a_2	01

By considering every node in the c-structure and f-structure and filtering out those that are the same for every analysis, one essentially obtains all discriminants. If, in spite of computing all discriminants, several analyses are left but no discriminants, then, disregarding marginal cases like those discussed in footnote 3, there must be a spurious ambiguity in the grammar and the analyses must be identical.

However, the annotator usually does not need to use all discriminants in the disambiguation process. In fact, in many cases just a few discriminant choices are needed to select the correct analysis amongst many. There is often considerable redundancy, because many discriminants are not independent of others. In order to make the annotator’s choices easier, it is therefore interesting to at least rank and perhaps also filter the discriminants that are presented to the annotator. Annotators will choose those that are the easiest and most useful to them. Our system keeps track of which discriminants are chosen. With this information, the display can be optimized so that, for instance, discriminants which are often chosen can be displayed first, and those that are not needed can be hidden from the display. Much work is still to be done in this area since it must be based on considerable testing in actual practice.

We have developed a toolkit that computes all discriminants and which is a testbed for optimizing their display. XLE-Web is a web-based interface to XLE with packed c- and f-structures and discriminants. The LFG Parsebanker is like XLE-Web, but also stores analyses and discriminant choices, and supports search in the stored analyses. For further details on this work, we refer to earlier publications (Rosén, Meurer, and De Smedt, 2005; Rosén et al., 2005; Rosén, De Smedt, and Meurer, 2006).

We currently display lexical and morphological discriminants first for several reasons. It has been pointed out that lexical ambiguities are often easier to decide on than others (Van der Beek et al., 2002; Oepen et al., 2004). Annotator decisions on lexical ambiguities also tend to be very reliable decisions, since they require little knowledge of the grammar. Decisions on lexical ambiguities are likely to be

safer than decisions on syntactic ambiguities because lexical and morphological discriminants contain such a small amount of information. Furthermore, decisions on lexical ambiguities are highly likely to be reapplicable on reparsing with a new version of the grammar, since part of speech changes and changes in morphological analysis will be rare.

With respect to syntactic ambiguities, different branchings are very intuitive (at least for linguists) and require little knowledge of the grammar. In many cases, branchings are quite independent of the grammatical theory used. For these reasons, we present both constituent and rule discriminants to the annotator.

Although not every discriminant is equally easy to decide on, the human disambiguator usually has enough choices of where to begin disambiguation that this does not really matter. Even though discriminant choices can be made independently, the discriminants themselves are not always independent. Choices also normally cause the resolution of other, dependent local ambiguities, making the disambiguation process even more efficient. Furthermore, a discriminant's applicability does not depend on the grammar, but only on the structures, so that discriminants can often be reused in an incremental parsebanking approach.

Discriminants can be exploited in various ways. The first and foremost application is in efficient manual disambigation to supplement the automatic parsing of a corpus, an approach also known as parsebanking. Parsebanking offers quality benefits over the manual construction of a treebank, including the avoidance of formal errors, consistency within the treebank and consistency with a grammar.

Another use of discriminants is in stochastic parse disambiguation. This approach uses properties of c- and f-structures as feature functions to train a stochastic parse ranking model (Riezler et al., 2002). XLE has property templates that can be used for this purpose. We have done experiments using our discriminants instead of the property templates. Preliminary testing of these two approaches has provided results that are better for discriminants than for property templates (Oepen et al., 2007).

6 Conclusion

In creating discriminants for LFG grammars, we have been guided by two important design principles. One principle is that enough discriminants must be computed to distinguish between all analyses. This means that all nodes in both c-structures and f-structures must be examined for possible discriminant candidates. The other main principle is that all distinctions must be represented in such a way that an annotator can easily relate them to words in the string. This ensures that disambiguation can be achieved quickly and efficiently.

Another important consideration has been our objective of making our methodology language and grammar independent. Our independence from particular languages and grammars follows from our approach which only builds on formal properties of representations. It would be possible to extend our design of LFG

discriminants to other projections. Although the Norwegian grammar has an MRS projection, and discriminants could be calculated on MRS properties, we have chosen not to do so. Since all LFG grammars have both c-structures and f-structures, complete disambiguation on these levels will be possible for any grammar and language.

One consequence of computing discriminants for all distinctions between representations is the large number of resulting discriminants. Often, however, individual structural differences are not independent of other differences. Rather than trying to eliminate some redundant discriminants by exploiting language specific interdependencies in their computation, we prefer to handle redundancy in their presentation. We have begun work on discriminant presentation in the context of the LFG Parsebanker, but this will be the focus of future research on how annotators use the tool. With the help of the LFG Parsebanker, the discriminants make it feasible to create large parsebanks for languages that have a broad coverage LFG grammar, something that until now has been impossible in practice because of the difficulty of disambiguating.

References

Butt, Miriam, Helge Dyvik, Tracy Holloway King, Hiroshi Masuichi, and Christian Rohrer. 2002. The Parallel Grammar project. In *Proceedings of COLING-2002 Workshop on Grammar Engineering and Evaluation, Taipei, Taiwan.*

Carter, David. 1997. The TreeBanker: A tool for supervised training of parsed corpora. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 598–603, Providence, Rhode Island.

Maxwell, John and Ronald M. Kaplan. 1993. The interface between phrasal and functional constraints. *Computational Linguistics*, 19(4):571–589.

Oepen, Stephan, Dan Flickinger, Kristina Toutanova, and Christopher D. Manning. 2004. LinGO Redwoods, a rich and dynamic treebank for HPSG. *Research on Language & Computation*, 2(4):575–596, December.

Oepen, Stephan, Erik Velldal, Jan Tore Lønning, Paul Meurer, Victoria Rosén, and Dan Flickinger. 2007. Towards hybrid quality-oriented machine translation. On linguistics and probabilities in MT. In *Proceedings of the 11th International Conference on Theoretical and Methodological Issues in Machine Translation*, Studies in Informatics. University of Skövde.

Riezler, Stefan, Tracy Holloway King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell, and Mark Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL'02).*

Rosén, Victoria, Koenraad De Smedt, Helge Dyvik, and Paul Meurer. 2005. TREPIL: Developing methods and tools for multilevel treebank construction. In Montserrat Civit, Sandra Kübler, and Ma. Antònia Martí, editors, *Proceedings of the Fourth Workshop on Treebanks and Linguistic Theories (TLT 2005)*, pages 161–172.

Rosén, Victoria, Koenraad De Smedt, and Paul Meurer. 2006. Towards a toolkit linking treebanking to grammar development. In *Proceedings of the Fifth Workshop on Treebanks and Linguistic Theories*, pages 55–66.

Rosén, Victoria, Paul Meurer, and Koenraad De Smedt. 2005. Constructing a parsed corpus with a large LFG grammar. In *Proceedings of LFG'05*, pages 371–387. CSLI Publications.

Van der Beek, Leonoor, Gosse Bouma, Robert Malouf, and Gertjan Van Noord. 2002. The Alpino dependency treebank. In *Computational Linguistics in the Netherlands (CLIN) 2001*, Twente University.