Treebanks as Parsed Corpora: An LFG Approach

Victoria Rosén, Paul Meurer, Koenraad De Smedt

University of Bergen and Uni Digital

CLARA Treebanking Course, December 13–14, 2010

Outline

The INESS Treebanking Infrastructure

INfrastructure for the Exploration of Syntax and Semantics (INESS): a specialized infrastructure for linguistic research

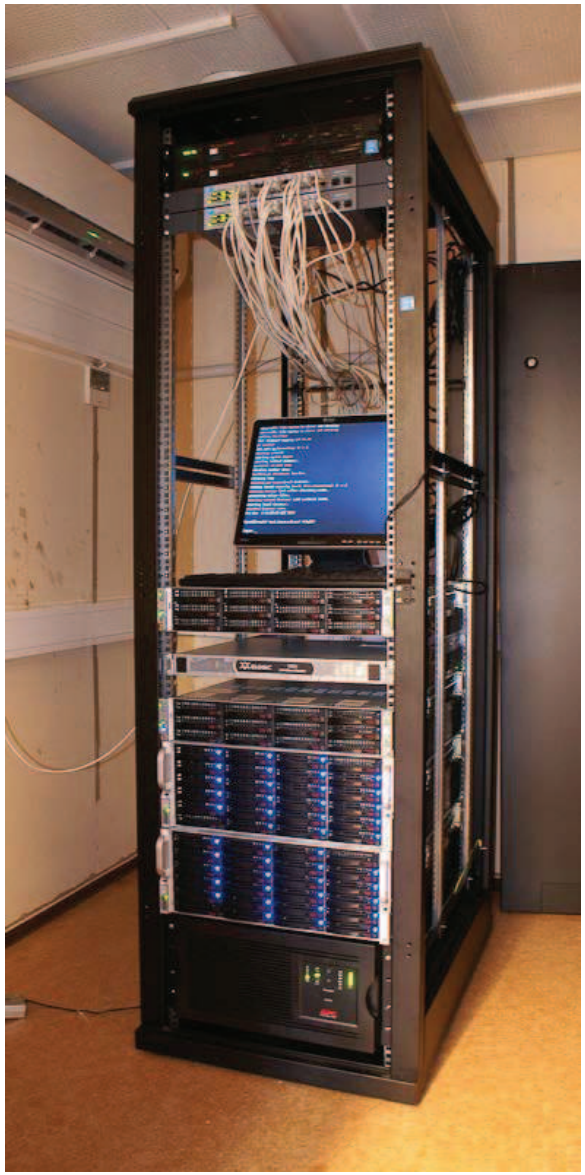
- Project 2010–2015 funded by the Research Council of Norway
- Led by University of Bergen, with national and international partners
- Based on previous projects LOGON and TREPIL (TREebank PILot project), which developed a parsebanking methodology and an advanced toolkit: the LFG Parsebanker

Project plan

- Develop the LFG Parsebanker further with focus on modularization, scalability and performance
- Integrate functionality for dependency treebanks and parallel treebanks
- Develop a 500 million word Norwegian treebank based on NorGram
- Host existing treebanks for other languages and in other formalisms (German TIGER treebank, Wikipedia treebank from Powerset etc.)
- Give interested parties the opportunity to develop their own treebanks using the INESS infrastructure
- Set up an HPC cluster with high capacity processing, storage and connectivity to host the treebanks

INESS will be integrated into the European CLARIN network.

HPC cluster



The INESS phase 1 computing and server system:

96 CPU cores (or hardware threads)

40 TB disk space

384 GB RAM

40 Gigabit per second internal networking (i.e. 40 times faster than Gigabit Ethernet)

Lexical-Functional Grammar

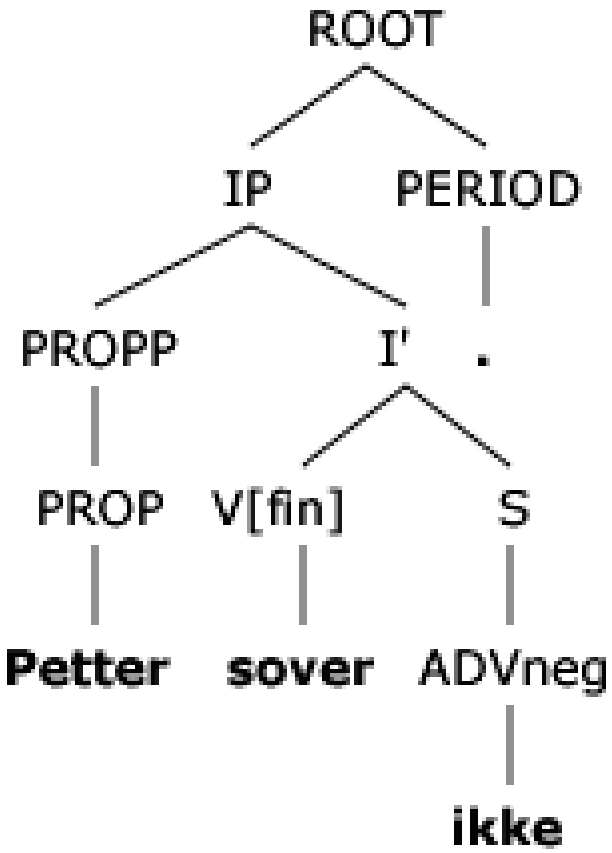
All LFG grammars have at least two projections:

1. c-structure (constituent structure, i.e. phrase structure tree)
2. f-structure (functional structure, i.e. attribute-value matrix)

Phrases are useful units in treebank applications

Deep syntax is also useful for treebank applications

C-structure and F-structure for *Petter sover ikke.*




```
PRED       'sove<[1:Petter]>NULL'
TNS-ASP  4 | TENSE pres, MOOD indicative |
TOPIC      | PRED   'Petter'
           | NTYPE  | NSEM 10 | COMMON count |
           |       8 | NSYN proper |
           | GEND  7 | NEUT -, MASC +, FEM - |
           | REF +, PERS 3, NUM sg, DEF +,
         1 | CASE nom
ADJUNCT { 24 | PRED      'ikke'
          | ADV-TYPE neg | }
SUBJ       [1:Petter]
VTYPE main, VFORM fin, STMT-TYPE decl,
MAIN-CL +
0
```

NorGram: An LFG grammar for Norwegian

NorGram is a large computational grammar for Norwegian Bokmål and Norwegian Nynorsk

First developed in the NorGram project led by Helge Dyvik

Further developed in other projects at the University of Bergen and Uni Digital

Size:

- 231 rules
- 5807 states
- 116716 arcs
- 602329 disjuncts

Preprocessing component

1. Morphological analyzer based on a lexicon with:
   - 140,000 base forms
   - 1,400,000 inflected forms
2. Compound analyzer:
   - Derives segmentations using regular expressions over strings and morphosyntactic features
   - Ranks them by number of segments and other heuristic criteria
3. Named entity recognizer based on constraint grammar

Overview

The LFG Parsebanker: A treebanking toolkit supporting LFG parsebanks

Main modules:

- LFG treebank module: Treebanking system for LFG analyses with a relational database backend (MySQL); versioning; multiple annotators
- Discriminants: Interactive manual disambiguation based on discriminants
- XLE-Web: Web interface for interactive parsing with LFG grammars using XLE, graphical display of parse results
- LFG Search: Reimplementation of TIGERSearch (search engine for dependency treebanks) to include support for c- and f-structures
- Parallel parsebanking extension

Discriminants: Motivation

The use of linguistically motivated handwritten grammars in applications is dependent on the availability of good disambiguation techniques.

Stochastic parse ranking is an obvious choice for automatic disambiguation; however, it is dependent on training based on a gold standard (corpus of disambiguated sentences).

Efficient techniques for manual disambiguation have been based on the concept of discriminants.

We extend these techniques to LFG grammars with a novel design and implementation.

Principles of discriminant-based disambiguation

Average sentences can have many possible analyses (most of which we are unaware of).

Carter (1997) realized the need for efficient disambiguation. He found that a few lexical or structural properties are often sufficient to distinguish the one intended analysis from many other analyses.

Such properties are e.g. different word senses, different bracketings, etc.

He calls these properties discriminants.

A similar approach is used in Alpino (Van der Beek, 2002) and LinGO Redwoods (Oepen et al. 2004).

Carter's TreeBanker

A graphical interface eliciting and recording manual discriminant choices (word senses, bracketing, rules)

Show me the flights to Boston serving a meal

- serve = fly to?
- serve = provide?
- flights to Boston?
- show –to Boston?

Efficiency in the TreeBanker

What is the earliest flight that has no stops from Washington to San Francisco on Friday?

yields 154 analyses and 318 discriminants, but only two choices are sufficient to narrow this down to two correct analyses

Degree to which discriminants are user-friendly (for the annotator):

constituents > semantic triples > word senses > sentence type > grammar rules used

Alpino (2002)

Van der Beek et al. built a large dependency treebank with the Alpino analyzer (HPSG).

They use Carter's rules, but different computation and ranking of discriminants.

Lexical discriminants are always presented to the annotator first because they claim lexical ambiguities are easiest to handle.

Non-lexical discriminants are ranked according to their discriminative power (sum of parses that will be excluded when marked bad and parses excluded when marked good).

LinGO Redwoods (2004)

Oepen et al.: LinGO HPSG grammar, Redwoods treebanking environment, [incr tsdb()] profiling tool

A dynamic treebank as a testbed for grammar development

Disambiguation often leads to revisions of the lexicon and grammar, improving coverage.

The treebank may then be reparsed with a new version of the grammar, yielding a new treebank version.

Discriminant choices from a previous version can be reapplied to disambiguate the new parses.

LFG discriminant design

How can discriminants be designed and implemented for LFG grammars?

- How can we find all possible distinctions between analyses?
- How can they be made recognizable to the annotator?

Four types of discriminants

1. Lexical discriminants
2. Morphological discriminants
3. C-structure discriminants
4. F-structure discriminants

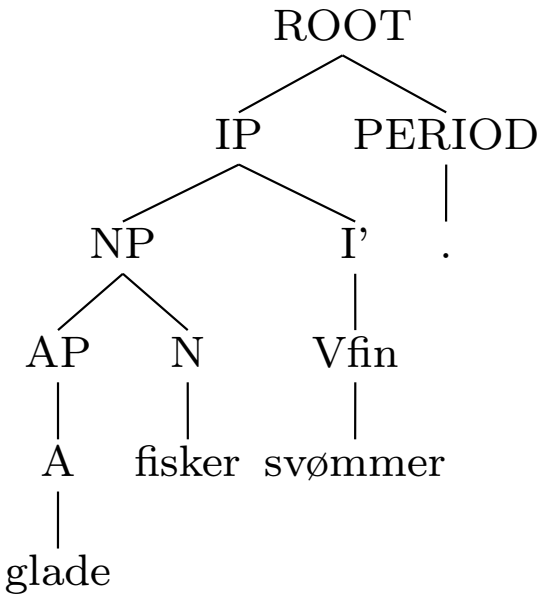
Lexical ambiguity

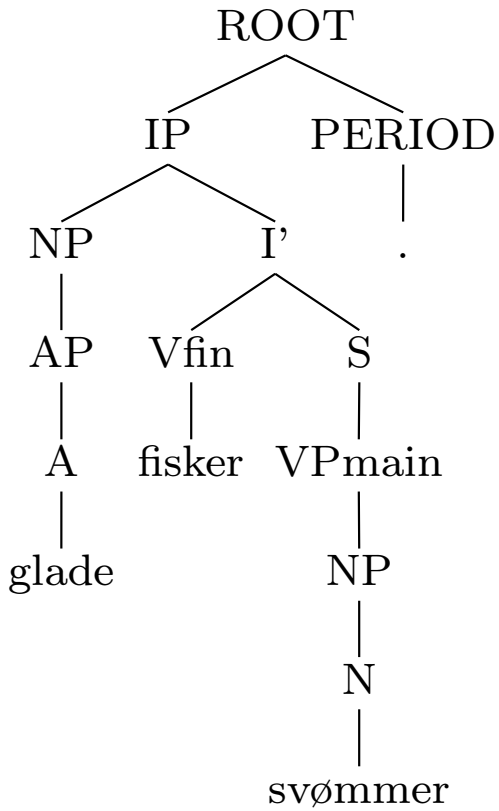
Lexical ambiguities are the easiest to disambiguate.

Two types of discriminants which aid in resolving lexical ambiguities:

1. Lexical discriminants
2. Morphological discriminants

Lexical discriminants





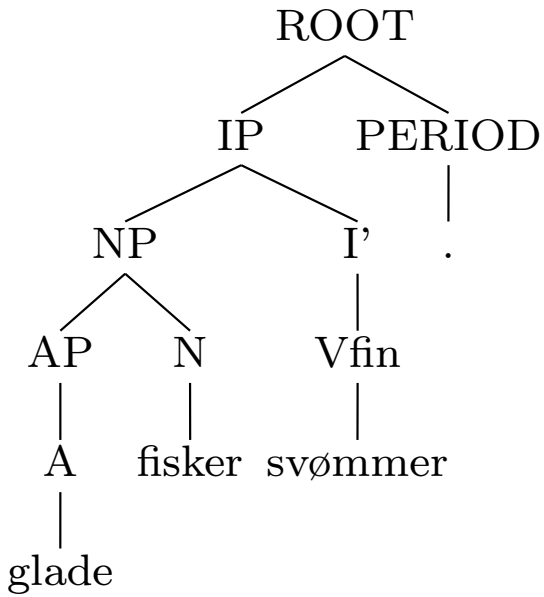
Lexical discriminants

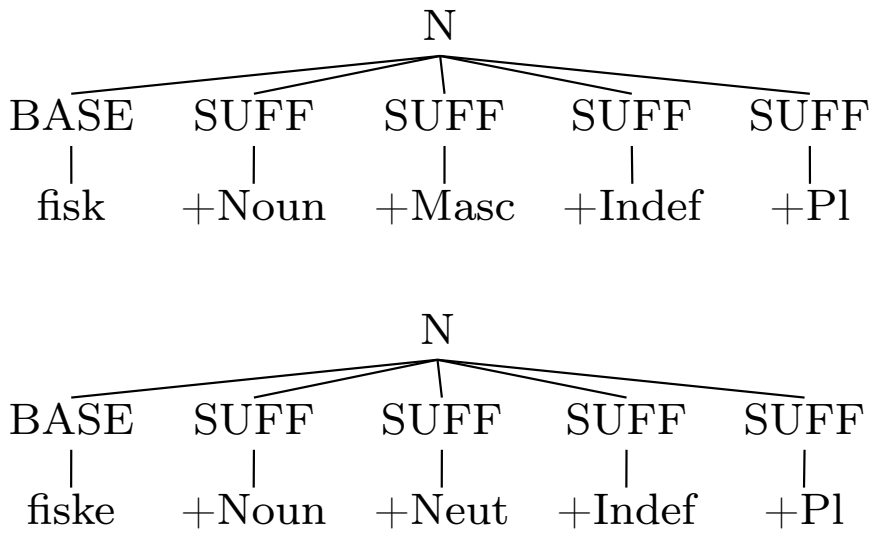
A lexical discriminant is a word form with its part of speech (or preterminal node label)

Representation of lexical discriminants:

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

Morphological discriminants





Morphological discriminants

A morphological discriminant is a base form with the tags it receives from morphological preprocessing. (Note: Words without morphological features are represented by the wordform itself.)

Representation of morphological discriminants:

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

Syntactic ambiguity

Two types of discriminants which aid in resolving syntactic ambiguities:

1. C-structure discriminants
2. F-structure discriminants

C-structure discriminants

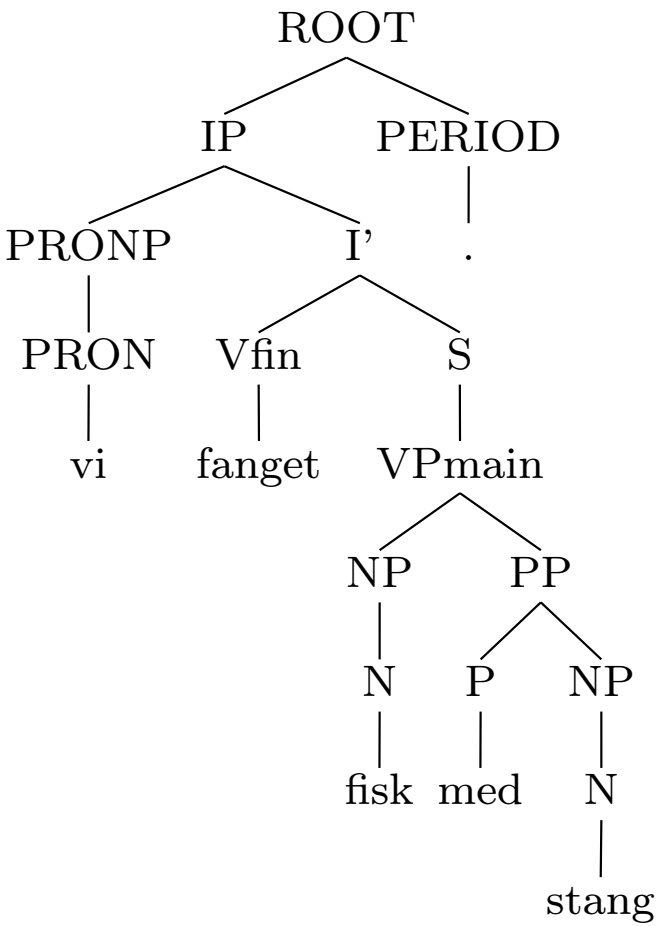
C-structure discriminants are important for the disambiguation of syntactic ambiguities, such as PP attachment.

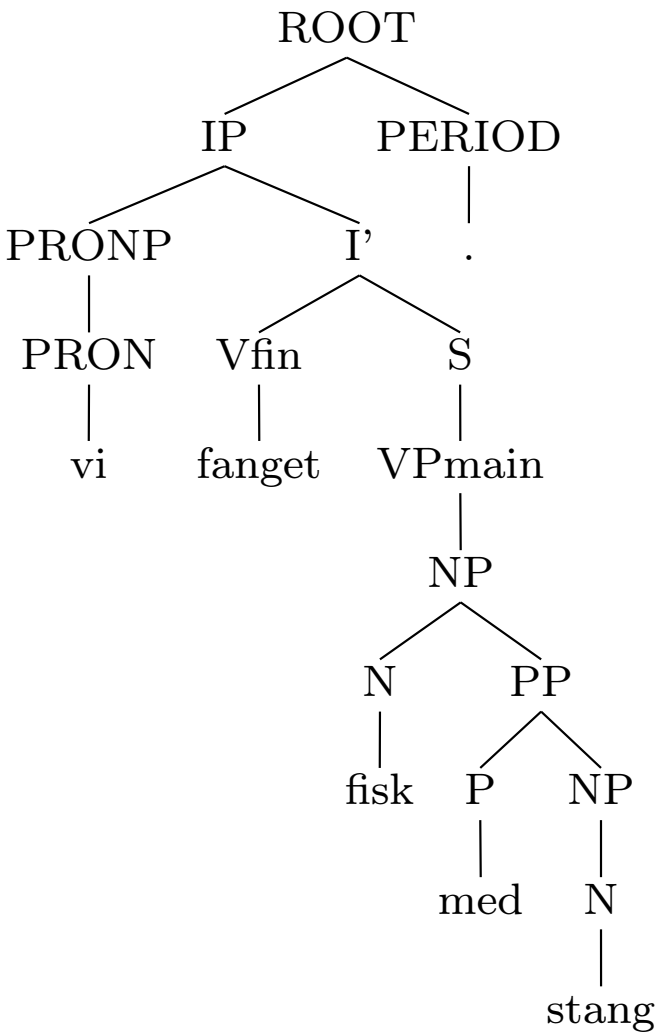
The idea is to pick out an elementary local property of a tree.

These pieces are minimal subtrees: a mother node and her daughters.

In order to be recognizable in isolation, the minimal subtree must be linked to the substring that it dominates.

C-structure discriminants: examples





C-structure discriminants: examples

Unlabeled and labeled bracketing of the substring *fisk med stang*:

[[fisk] [med stang]]

[$_{VPmain}$ [$_{NP}$ fisk] [$_{PP}$ med stang]] [$_{NP}$ [$_{N}$ fisk] [$_{PP}$ med stang]]

Two subtypes of c-structure discriminants

constituent discriminants: fisk || med stang

rule discriminants: VPmain → NP PP NP → N PP

F-structure discriminants

Based on partial paths through f-structures

An f-structure discriminant is a minimal path through the f-structure

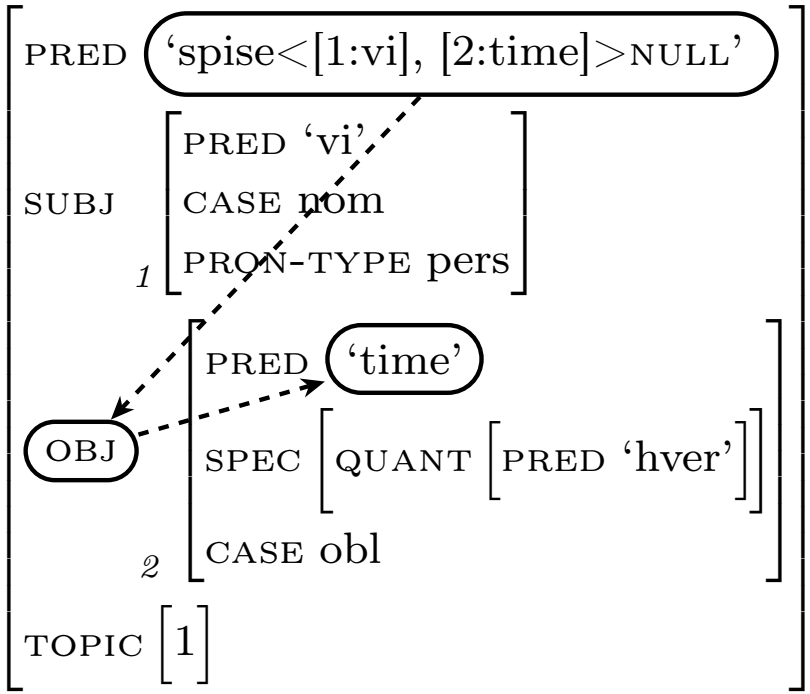
- from a PRED value to another PRED value, or
- from a PRED value to an atomic value

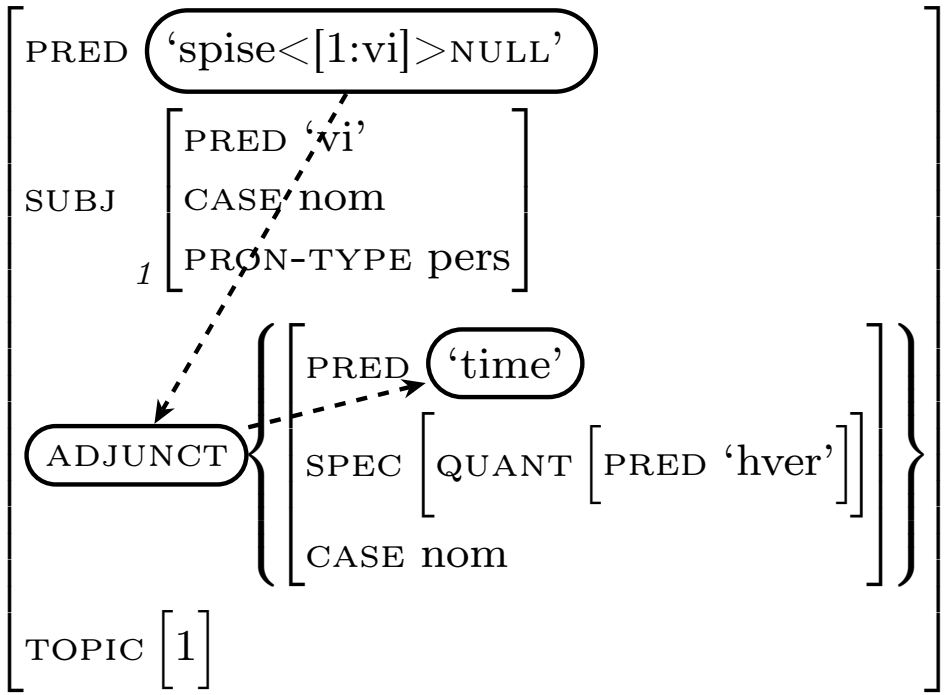
(a minimal path being one that does not cross any intermediate PRED values)

F-structure discriminants: examples

Vi spiser hver time.
we eat every hour
"We eat every hour."

F-structure discriminants: examples





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

F-structure discriminants

The path may also go from a PRED to an atomic value

Vi liker barn.
we like child
"We like child/children."

'barn' NUM sg
'barn' NUM pl

Discriminant anchors

All four discriminant types relate linguistic properties to words in the string to make it easy to recognize the desired properties.

However, the same word or substring may appear more than once in a sentence.

De store fisker spiser de små fisker.
the/you/those big fish.N/fish.V/fishing eat/eater the small fish.N/fish.V/fishing
"The big fish eat the small fish."/"The small fish, the big fish eat."/"Those big fish eat the small fish."/etc.

Discriminant anchors

De store fisker spiser de små
the/you/those big fish.N/fish.V/fishing eat/eater the small
fisker.
fish.N/fish.V/fishing
"The big fish eat the small fish."/"The small fish, the big fish eat."/"Those big fish eat the small fish."/etc.

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

Each discriminant is linked to string position through an anchor.

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

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