MASSIVE DISAMBIGUATION OF LARGE TEXT CORPORA WITH FLEXIBLE CATEGORIAL GRAMMAR

Ton van der WOUDEN (CELEX/INL)
Dirk HEYLEN (INL)

INL
Postbus 9515
2300 RA Leiden
The Netherlands

ABSTRACT

A new method of automatic lexical disambiguation of big texts is described, using recent proof-theoretical results from the theory of categorial grammar.

0. Introduction

The Institute of Dutch Lexicology (INL), sponsored by the Dutch and the Belgian governments, consists of two departments. One of the tasks of the Thesaurus, one of the departments, is to build a database for lexicological research. This database is a, potentially infinite, set of texts. The aim is to supply a representative and, if possible, complete overview of contemporary (since 1970) standard Dutch. In order to access the material, an efficient database architecture and application software were developed. At this moment (February 1988) the INL corpus is the most representative corpus of the Dutch language; it contains over 45 million tokens, over 800,000 word types.
The INL lexical database is not an aim in itself: it is meant to be a tool for specific projects. One of the purposes of the database is to form the raw input for a new generation of dictionaries. The database is growing, and it is not realistic to go on increasing it without making it possible to use such a rich source of information. Therefore, a short time goal is to make all the words in the database available for research purposes, on the one hand by making efficient and powerful application software, on the other hand by enriching the material. Automatic morphological analysis has now been carried out and the results will soon be incorporated into the database. One level higher, we are interested in the syntactic analysis of the sentences of the corpus. One should not think of an on-line parser only. For the process of lemmatization an effective disambiguation procedure is necessary as well. To this end parsers are being developed that will soon be tested on the corpus. As was the case for the morphological analyzer, the syntactic parser is an implementation of a categorial calculus. The construction of and philosophy behind the Lambek categorial parser we use for the disambiguation and syntactic analysis is the topic of this paper[1].

1. A note on ambiguity in Categorial Grammars

Each linguistic model or framework whatsoever is confronted with the problem of ambiguous lexical type assignments, a phenomenon inherent to NL. Whatever way one deals with it as far as representation is concerned and whatever neat solutions one comes up with, the fact remains that (1) the phenomenon will not disappear, but (2) the explosions it gives rise to will cause (often irreparable) damage to (otherwise) neatly conceived syntactic parsers or analyzers. Categorial grammars, abiding by the centricity of the Lexicon, may seem by nature to be the first victims of this phenomenon. Some categorialists[2] try to circumvent the problems by imposing inherently unmotivated constraints on otherwise rigidly defined flexible calculi. Another way to go about, however, is to take a closer look at some of the restrictions the calculus imposes indirectly, i.e. at some of the invariants that come along naturally, but may remain unnoticed at first sight[3]. Interesting invariants may act as greedy scissors, pruning away many of the useless branches of the search tree. Categorial grammars encode all syntactic information in the lexicon. The effect of this strategy on the presence of ambiguities can be gathered if one would take an ordinary phrase structure grammar and turn it into a categorial one. What happens is that for every category in the PS grammar one gets a set of categories in the Categorial grammar. On the avarage, the number of new categories equals the number of occurrences of the old category in the PS rules. A lexical element that is not at all ambiguous as far as syntactic category assignment is concerned, in PSG, will almost certainly become ambiguous in CG. Still, we claim that effective, i.e. fast, disambiguation, is possible with CG. The rationale behind this claim is that effective disambiguation does not depend as much on the degree of ambiguity, but first and foremost on the nature of the disambiguation method. Whereas ambiguity is damaging to classical parse procedures because there are no intrinsic properties of the system that can deal with it, almost the reverse is true of categorial parsers when full benefit is made of their defining characteristics. In order to appreciate these statements, the best thing to do is look at a specific implementation of this idea.

2. The Lambek calculus

In this section we would like to present a categorial reduction system which is analogous to the implicational fragment of propositional logic. We will present it as a calculus, and will limit ourselves to the formal description, thus ignoring semantic interpretation (which is not immediately relevant for our purpose at hand).

Some definitions

Let BASCAT be a finite set of atomic categories and CONN a finite set of category forming connectives. Then CAT (the set of all categories) is the inductive closure of BASCAT under CONN, i.e. the smallest set such that (i) BASCAT is a subset of CAT, and (ii) if X, Y are members of CAT and | is a member of CONN, then (X|Y) is a member of CAT.

So one could take BASCAT to be {S, N, A, T, P} and CONN {/, \, *} (these are called right division, left division and product, respectively). Some of the members of CAT are: {N, (N\S), ((N/N)*T), (S/(P\(N/S))),...}.

A complex category (X|Y) consists of three immediate subcomponents: X and Y, which are themselves categories, and the connective. When the connective is '/' or '\', the complex category is a functor. Functor categories are associated with incomplete expressions: they will form an expression of category Y (result) with an expression of category X (argument)[4]. In the case of right division, the argument has to be found to the right of the functor category, whereas in the case of left division, the argument has to be found to the left[5]. The product connective '*' is to be interpreted as a concatenation operator, i.e. a product category (X*Y) is to be associated with an expression which is the concatenation of an expression of category X and an expression of category Y in that order.

Reduction rules

A specific categorial grammar is characterized by the choice of basic categories and connectives on the one hand, and on the set of reduction rules on the other. The system of reduction rules says how categories can be combined to form larger constituents. The application rule which combines a functor with domain X and range Y with a suitable argument of category X to give a Y, is only one of the possible reduction rules. Instead of taking a set of reduction laws as primitive axioms, we will investigate the categorial reduction system as a calculus, where the reduction laws can be considered theorems that follow from a set of axioms and a set of inference rules. Next we will see that the parsing of a syntagm is really the same thing, in other words, attempting a proof for a theorem.

Sequents

Before we define the axioms and inference rules of the calculus, we need to define the notion of sequent[6].

A *sequent* is a pair (G,D) of finite (possibly empty) sequences $G = [A_1, \ldots A_n]$, $D = [B_1, \ldots, B_n]$ of categories. For categorial L-sequents, we require G to be non-empty and n=1. For the sequent (G,D) we write G => D. The sequence G is called the antecedent, D the succedent. For simplicity square brackets and comma's are often left out.

Axioms and inference rules

(1) The *axioms* of L are sequents of the form X => X.

(2) Inference rules of L: X, Y and Z are categories, P, T, Q, U, V are sequences of categories, where P, T and Q are non-empty.

```
[/R]   T => Y/X if T,Y => X
[\R]   T => Y\X if Y,T => X
[/L]   U,Y/X,T,V => Z if T => Y
                     and U,X,V => Z
[\L]   U,T,Y\X,V => Z if T => Y
                     and U,X,V => Z
[*L]   U,X*Y,V => Z if U,X,Y,V => Z
[*R]   P,Q => X*Y if P => X and Q => Y
```

Together, axioms and inference rules define the theorems of a categorial calculus. Suppose we have a sequent S, to find out whether it is a theorem or not we have to apply several of the inference rules above till nothing but axioms remain. As one may have noticed, all these rules involve the removal of a connective in some category. Let's paraphrase the [/L] rule by way of example. It says: to find out whether a sequent with some functor category Y/X is a theorem, identify a sequence of categories that follow this category, and see whether *Y => the identified sequence* is a theorem, and *what preceded the category + X + what followed the sequence => old succedent* is a theorem.

In the following example we present a proof with the relevant category printed in bold and the *identified sequence* underlined.

```
a/b, d/(e/(f/a)), d, e, f  =>  b    [/L]
  d => d                          [AXIOM]
     a/b, e/(f/a), e, f => b        [/L]
     e => e                       [AXIOM]
        a/b, f/a, f => b            [/L]
        f => f                    [AXIOM]
           a/b, a => b              [/L]
           a => a                 [AXIOM]
           b => b                 [AXIOM]

  QED
```

If we could find an efficient automatic decision procedure that would tell us whether a certain sequent is either a theorem or not, then we would have an efficient parser as well. The idea being, that the succedent represents something like a sentence (the categories of the words that make it up) and the antecedent the **S** (sentence) category. In the next section we will discuss an implementation of the decision procedure.

3. The Theorem prover, alias parser

An algorithm to prove a theorem could go as follows.

Given: a sequent with n categories: n-1 in antecedent, 1 in succedent.

Start at the the first category of the succedent. If this is a functor, pick the relevant inference rule that will eliminate the connective. If the rule tells you to identify a part of the sequent to one of the sides of the category, then first take this to be one category. See whether you can prove the resulting sequent(s) (the sequent(s) in the if-part of the inference rule). If the identification does not yield a result (i.e. in recursively calling the procedure, the bottom of *only axioms remaining* is not reached), then take two categories and see if this does the trick. Continue adding categories until you have a proof or there are no categories left. In the latter case, nothing is lost yet, because one could also have taken the second, or third functor to start the proof with. If in the end there are no more functors left to start the elimination with, then the theorem cannot be proven and one can even say that it is false[7].

Clearly, this procedure might take some time to decide on the validity of a sequent. One might hope that theorems are proven rapidly, but when the sequents are false, a lot of work has to be done. Fortunately enough, there is a simple way to prune away some branches of the search tree that are guaranteed to lead to failure. There is a necessary formal condition that holds of valid theorems which is easy to detect. If a sequent does not have this formal characteristic, it cannot be a theorem. Even if the inputted sequent does have the required characteristic, in the process of proving, there will be a lot of subproofs that need not be carried out because they will fail immediately.

This formal characteristic or invariant is known as van Benthem's Count, or Count for short. It counts the number of positive (range) and negative (domain) occurrences of a basic category X in an arbitrary category, basic or complex. It may be defined as follows.

```
count(X,X) = 1, if X is a member of BASCAT
count(X,Y) = 0, if X,Y members of BASCAT,
                X <> Y
count(X,Y/Z) = count(X,Z) - count(X,Y)
count(X,Y\Z) = count(X,Z) - count(X,Y)
count(X,Y*Z) = count(X,Y) + count(X,Z)
```

Generalized to sequences of categories, the X-count of a sequence, X being a category, is the sum of the X-counts of the elements in the sequence.

$$count(X,[Y_1,\dots,Y_n]) = count(X,Y_1) + \dots + count(X,Y_n)$$

It was proven by Van Benthem (1986) that the Count function is an invariant over derivations[8]. This means that no sequent is a theorem if the count values of the antecedent differ from the count values of the succedent for any basic category. The following figure shows how the count values for the category (PP/(NP\S)) can be computed for each of the basic categories S, NP, N, AP and PP.

	S	NP	N	AP	PP
c (PP/(NP\S))	1	-1	0	0	-1
- PP	0	0	0	0	1
+ (NP\S)	1	-1	0	0	0
- NP	0	1	0	0	0
+ S	1	0	0	0	0

To see the usefulness of this invariant take a noun phrase like **de groei van het haar** ('the growth of the hair'). Apart from de, all words in this NP are ambiguous. The Cartesian product of the ambiguities gives 12 different combinatory possibilities:

```
(N/NP),N,(NP/PP),(N/NP),NP
(N/NP),N,(NP/PP),(N/NP),(N/NP)
(N/NP),N,(NP/PP),(N/NP),N
(N/NP),N,(NP/(N\N)),(N/NP),NP
(N/NP),N,(NP/(N\N)),(N/NP),(N/NP)
(N/NP),N,(NP/(N\N)),(N/NP),N
(N/NP),(NP\S),(NP/PP),(N/NP),NP
(N/NP),(NP\S),(NP/PP),(N/NP),(N/NP)
(N/NP),(NP\S),(NP/PP),(N/NP),N
(N/NP),(NP\S),(NP/(N\N)),(N/NP),NP
(N/NP),(NP\S),(NP/(N\N)),(N/NP),(N/NP)
(N/NP),(NP\S),(NP/(N\N)),(N/NP),N
```

To figure out whether this phrase is a noun phrase, one would have to try to build a (NP) parse tree for each of these twelve possible combinations of category assignments. Using the Count invariant, however, one knows beforehand that one and only one of these combinations (given in bold face) could possibly be parsed as a noun phrase, so that parsing itself becomes superfluous in this case. The following figure shows the count values for the correct assignment.

N/NP	0	1	-1	0	0
N	0	0	1	0	0
(NP/(N\N))	0	-1	0	0	0
(N/NP)	0	1	-1	0	0
N	0	0	1	0	0
+	0	1	0	0	0
NP	0	1	0	0	0

The reader can verify for himself that none of the other combinations satifies the count invariant.

4. The implementation

It is obvious that the procedure just presented is a perfect means to lay hands on the ratios of the frequencies of lexically ambiguous words, given a corpus and a lexicon with categorial information. So, in order to derive these figures for the words in the CELEX database, sentences of the INL corpus are inputted in a cascade of disambiguating modules. The implementation of this Lambek-Gentzen disambiguator is straightforward as it involves only simple matchings and list-manipulations. The role of the disambiguator in the process of disambiguating the L corpus can be read off from the following figure.

```
    CORPUS          : set of sentences
 => sentence selection
    SENTENCE (1)    : list of words
 => category assignment / lexical lookup
    SENTENCE (2)    : list of words +
                      categories
 => generate combinations
    COMBINATIONS    : list of categories
 => test combinations
    (1) Count
    (2) Parse / prove
    RESULT          : grammatical lists
                            of categories
```

Given a corpus sentence, the syntactic categories of all the words it contains are looked up in a parsing lexicon derived from the lexical database. When all combinations of categories have been computed, each is tested by the Count module to reduce the number of possible combinations of initial category assignments. In the most successful case, this reduction produces only one possible combination, implying that all lexical material in this sentence is disambiguated. In most other cases, only a small percentage of the original number of possible combinations of lexical assignments is left over; these are handed over to the Gentzen Proof Machine which will find out which of the remaining assignments fail to combine to a grammatical sentence.

Notes

1. Much of the work described here is based on research by Michael Moortgat. See e.g. his (1987a, 1987b, 1988).

2. e.g. Wittenburg (1987), Steedman (1987).

3. Instead of theorems deducible from the calculus they are often facts that can be proven of the calculus as such, outside the calculus (metatheorems in other words).

4. This combination is called application.

5. Notice that we will use the (argument connective result) notation, no matter what the directionality of the functor.

6. We will present the sequent calculus, which Lambek adapted from Gentzen's work on logic. See Lambek (1958).

7. Because of space limitations we will not attempt to show the validity of this procedure.

8. Proof omitted for space's sake.

Bibliography

J. van Benthem (1986) Categorial Grammar. Ch. 7 of **Essays in Logical Semantics**. Reidel, Dordrecht.

J. Lambek (1958) The mathematics of sentence structure. In: **Am. Math. Monthly** 65, 154-169. Reprinted in Buszkowski e.a. (eds.): **Categorial Grammar**. Benjamin's, Amsterdam (to appear).

M. Moortgat (1987a) Lambek Theorem Proving, INL-WP 87-04. In Van Benthem & Klein (eds.): **Categories, Polymorphism and Unification.**

(1987b) Generalized Categorial Grammar. To appear in F.G. Droste (ed.): **Mainstreams in Linguistics**. Benjamin's, Amsterdam.

(1988) **Categorial Investigations** (dissertation, to appear).

M. Steedman (1987) Combinators and Grammars. In Oehrle, Bach & Wheeler (eds.): **Categorial Grammars and Natural Language Structures.** Reidel, Dordrecht.

K.Wittenburg (1987) Predictive Combinators: A Method for Efficient Processing of Combinatory Categorial Grammars. In **Proceedings ACL 25.**