Computational grammars and Ambiguity : the bare bones of the situation

Max Copperman and Frédérique Segond

December 10, 1996

Abstract

Linguists have been writing grammars for many years. Writing a computational grammar for use in an application is a different enterprise from writing a grammar to explore linguistic theory. Often, applications require a very small number of analyses of a sentence. Mechanisms must be found to reduce ambiguity. Here we present several techniques for reducing lexical and syntactic ambiguity and for ambiguity resulting from multi-word- expressions.

1 From Linguistics to Language Engineering

Linguistics was first done on paper. In going from paper-and-pencil linguistics to computational linguistics to language engineering, new issues emerge. One thing in particular distinguishes computational grammars from "paper" grammars: a computational grammar must be completely and formally specified. It is not possible to simply assume the existence of a lexicon: the lexicon must be encoded. It is not possible to work only on a particular construction, unless one is willing to accept an inability to parse real texts.

This requirement becomes much more stringent when it is combined with a requirement of NLP applications: to deal with text as it occurs in the domain of interest. Lexicons typically must be large enough that they cannot be constructed by hand for each new application or new domain. And while particular domains do not use completely unrestricted language, the sublanguages used in many domains are complex enough to push the limits of linguistic theory, and to require rule sets of considerable complexity. Nowadays large-scale lexicons are available, which contain all the lexical ambiguities that lexicographers have encoded over the years. This changes both the grammar writing task and the lexicon building task, making lexicon engineering an important part of grammar writing.

In addition, if NLP applications are to be successful, the grammars must be readable, maintainable, extensible, and to a certain extent reusable. This puts a limit on their complexity that may be at odds with the effort to describe a particular sublanguage in detail. A further consideration is that the complexity of a grammar is tied to the level of detail of description of the language. It is well known in software engineering that the more complex a piece of software is, the harder it is to read, maintain, and modify—it becomes brittle. Like software, as the complexity of a grammar increases, the grammar becomes more brittle.

Finally, a most severe requirement is that NLP applications typically require a very small number of analyses, sometimes one. Computerized syntactic analysis can result in dozens of analyses for a sentence that is unambiguous to a human. Some mechanisms must be found to trim the analysis set down to an acceptable size. While there can be arbitrary processing after syntactic analyses, this processing typically requires enumeration of the analyses, which can make the post-processing even more expensive (especially because, depending on the application, post processing may include a stage where it creates more output).

	LING	COMP	NLP
theoretically motivated	X		
completely specified		X	X
handles real text			X
returns few analyses			X
maintainable, extensible			X

Table 1: A comparison of particular requirements of several kinds of grammars: LING denotes a theoretical linguist's pencil-and-paper grammar, COMP denotes an arbitrary computational grammar, and NLP denotes a computational grammar used in an NLP application.

Furthermore, the need for a small number of analyses may pit processing expense against correctness. In the limit, NLP is AI complete. In the face of this fact, it is reasonable and appropriate to settle for a "best guess" that can be arrived at via available techniques and heuristics. Wisdom lies in choosing techniques and heuristics that give as good results as possible today, without precluding eventual improvements based on theoretical advances.

The first computational tools built for LFG were built by and for linguists, to aid them in performing linguistic investigation or in teaching linguistics ([KM93], [Ko96], [Ma96a], [Av96]). There has been a growing additional emphasis: as the fields of linguistics and computer science have reached the point where natural language processing (NLP) tasks become tractable, LFG computational tools are being applied to NLP tasks—LFG is being used for "language engineering". In this usage, LFG faces the requirements of NLP grammars.

We list a number of different kinds of ambiguities and suggest some possible techniques for handling some of them, keeping in mind that the application of a technique always has the potential to eliminate a valid analysis.

2 Ambiguity

Ambiguity is a general term whose precise definition depends a lot on the application one has in mind. For instance, if one has in mind machine translation, a Japanese sentence with a single analysis can be seen as ambiguous as soon as it has to be translated into French, because Japanese has no gender and uses no determiners ([BT96]). In this paper, we focus on ambiguity when it takes the form of multiple syntactic analyses.

2.1 Types of Ambiguity

Obtaining different analyses for an input sentence might have different causes: lexical, syntactic, semantic, pragmatic, etc., and the interaction of all these levels. We will concentrate on the first two, as well as on multi-word expressions (MWEs).

1. Lexical/morphological:
   (a) Words often fall into multiple part of speech categories. This leads to many analyses. For instance, consider the following French phrase: `voyant orange`. `Voyant` falls into three categories: noun, adjective, and verb (present participle), and `orange` may be a noun or an adjective. Because of this we get 3 valid NP structures. It is easy to picture the combinatorics when this particular NP is embedded in a longer sentence with other ambiguities.
   (b) Within a given part of speech, words can have multiple argument structures. This is the case for most verbs, but is not limited to verbs. For instance, the French word `fier` is an adjective. As such it has an entry with just a PRED. But like many adjectives in French, `fier` can also subcategorize for a PP. We will therefore get two analyses for a phrases `il était fier de son travail`: `((il était fier)`

(de son travail)) and (il était (fier de son travail)). Providing a word with multiple argument structures is necessary in order to get the correct analysis, but it is a source of ambiguity.

(c) Within a given part of speech, words can have multiple f-structures with distinct feature collections. For instance the word page in French is both masculine and feminine, meaning something different in each gender. As a consequence, the sentence les pages se dchirent facilement ("pages tear easily") will have two analyses.

2. Syntactic:
The most important phenomena of syntactic ambiguity is attachment of the adjuncts (PP attachment, adjective attachment, etc.), which results semantically in scope ambiguity. The sentence:

1 La mère autorise une sortie à l'enfant.

may be parsed with the PP à l'enfant as part of the verb subcategorisation (La mère (autorise (une sortie) (à l'enfant))), with it attached to the NP sortie (La mère autorise une (sortie à l'enfant)), or with it attached to the whole sentence (la mère autorise une sortie (à l'enfant)).

3. Multi-word Expressions:
Multi-word expressions cross the dimensions mentioned above, having impact in the lexical, syntactic, and semantic domains. We have examples such as bien que in French, which may be two independent words (good, which/that), but taken as a single expression, means although. As an MWE, it is syntactically idiosyncratic, in that the grammar will not combine bien with que to produce a conjunction, which is bien que's category.[1]

There are MWEs that are perfectly regular in their syntax and could be parsed by a grammar that did not take MWEs into account, but have an idiomatic meaning as well as a compositional meaning. This latter case we believe should have a single syntactic analysis, just as quantifier-scope-ambiguous sentences do, though they have multiple semantic analyses. While fascinating, semantic ambiguity is beyond the scope of this paper.

2.2 Disambiguations Methods

Before suggesting techniques to deal with some classes of ambiguities, we would like to look more closely at what writing a computational grammar implies.

Writing a computational grammar is a distinct enterprise from writing a grammar, with pencil and paper, that is processed through hand-simulation and introspection. As mentioned above, the computational grammar must be fully specified in some formalism, in our case LFG. In addition, the computer is relentlessly thorough: it finds unforeseen interactions, and requires that they be handled in full detail, whether they are of theoretical interest or not. In fact, a large proportion of the grammar engineer's time is spent dealing with unforeseen interactions. Some of these interactions are problematic.

Problematic interactions come in two types:

- Interactions that produce unforeseen results. For example, the sentence:

  2. dis le bien que tu penses.

  has the meaning "Tell me the good things you think.", and in the desired analysis, bien is a noun and que is a relative pronoun. But in the desired analysis of the sentence:

  3. fais le bien que tu manges.

  which means, "Do it even though you're eating", bien que is a conjunction meaning "even though". Suppose your grammar accounts for 2. Then it will produce an incorrect

[1] The grammar will not combine bien and que at all; they do not form a constituent. However, there are many syntactically idiosyncratic MWEs that do form a constituent. Their idiosyncracy lies in the fact that the category of the constituent so derived is not the appropriate category for the string considered as an MWE.

analysis of 3. But a native speaker would not foresee, even when adding the grammar rules to account for 2, that these rules will interact with the lexicon to produce an analysis of 3 that, roughly speaking, would mean "Do the good that you eat.".

- Interactions that block expected results. In order to avoid the analysis which gives `possible que le moteur démarre` as an NP in the sentence `il est possible que le moteur démarre`, we might constrain NPs to have a determiner. In doing so, we might not foresee that this will block the parsing of perfectly acceptable sentences such as: `particulier vend futon`.

2.3 When and How to Attack Ambiguity

Ambiguities can be dealt with before, during or after parsing. If an ambiguity is dealt with early, there is the possibility of losing the right analysis. If it is dealt with late, there is the computational cost of processing many analyses. The proper balance point in this tradeoff varies for different types of ambiguities, and there is no universal metric.

In an LFG framework, a grammar writer may add features in whatever way is necessary to eliminate an analysis, but this weakens any claims about what features in general mean. It complicates the grammar. And it severely weakens the cross-linguistic validity of f-structures, since the f-structures are full of features added for this purpose, which vary from language to language.

In what follows we suggest some techniques for handling lexical and syntactic ambiguities at different stages. Multi-word expressions cross these dimensions, and we suggest some approaches to them below.

2.3.1 Coping with lexical ambiguity

Lexical ambiguity is a major cause of multiple analyses, because many words and especially very common words are ambiguous. For instance `la` in French can be either a noun or a determiner. The probability that it occurs as a noun in technical texts is very low, and similar facts can be noted about other words with respect to technical texts.

A part-of-speech tagger takes these probabilities into account in a way that a (non-probabalistic) syntactic grammar does not. It is possible to tag a sentence prior to parsing it, and base the parse on the sentence as disambiguated by the tagger ([Hi94]). Unfortunately, due to the error rate of part-of-speech taggers, the benefits of this tagger-based disambiguation are offset by the introduction of parse errors due to mistagged words.

The idea that the lexical ambiguity can be reduced, for a particular genre, or for a particular corpus, by using a genre-specific or corpus-specific lexicon, remains attractive. However, the effort of entirely manually constructing a lexicon is too great to consider, especially when large-scale general lexicons are available on-line. Furthermore, a hand-constructed lexicon can hide a lack of generality in the grammar, because the interactions with the lexicon are overly constrained. But a specialized lexicon may be built from a general one.

We have successfully used a tagger to filter a lexicon for part-of-speech ambiguity in a particular domain. The key to reducing the effect of tagger errors is that the results are not used on a sentence by sentence basis, but accumulated across an entire corpus. That is, we tag the entire corpus and, after checking the possible tagging errors over the corpus, place into a specialized lexicon those <*word*, *part-of-speech*> pairs that appear in the tagged corpus. This lexicon will contain only the words forms together with the parts of speech that are relevant for the texts.

The steps can be expressed as follows:

1. Tag the corpus. Sort the tagged corpus and eliminate duplication. This gives a list of <*word*, *POS*> pairs actually found in the texts, (where *POS* is a part-of-speech label). For technical texts, there is a high probability that fewer parts of speech will be paired with a word in this list than in a large-scale dictionary.

2. Manually track the tagger errors. For instance, the tagger may mis-tag a word with a POS that no occurrence of that word in the corpus ought to be tagged with. This is not strictly an error, but rather a less-than-optimal reduction in lexical ambiguity. In any case, if a word appears in the tagged corpus with an unlikely POS, we just have to check the occurrences of the rare POS in the corpus.

   For example, `est` in French is a noun ("east") and a verb ("to be"). If it appears in the list for our technical text with the noun POS (the less probable), we locate the occurrence of `est` tagged as a noun in the tagged corpus and check if it is correct. While this does require manual intervention, it is neither difficult nor time-consuming.[2]

3. From a large-scale lexicon, for each pair (*w*, *POS*) on the list, extract those parts of the lexical entry for *w* that have part of speech *POS*. The resulting corpus-specific lexicon is the one the parser uses.

We are able to apply this method because we have access to the entire corpus—there will be no new words (which would be unknown to our specialized corpus). But words unknown to this lexicon could be looked up in the large-scale lexicon, and used with their full lexical ambiguity.

The result is a more appropriate lexicon for the corpus, and as a consequence, we have fewer undesirable analyses. Restricting a lexicon for a particular domain is a different activity from constructing a lexicon for a particular domain, and there is something to be learned from it.

2.3.2 Coping with syntactic ambiguity

The primary method of reducing syntactic ambiguity is to write more restrictive rules. For instance, we can eliminate one of the analyses of:

4. `Interest rates will drop to 7%.`

by disallowing attachment of sentence-final PPs to the sentence level. But this will be incorrect for some PPs:

5. `Interest rates will drop, according to government sources.`

In many cases it is possible to find a restriction that rules out exactly the undesired analyses. But this is not always possible. For example, examples 2 and 3, repeated here:

2 `dis le bien que tu penses`

3 `fais le bien que tu manges.`

are equivalent under all syntactic distinctions made by the grammar. We must choose between living with the ambiguity or failing to analyze one of them correctly.[3]

These choices are unavoidable: we choose to fail to analyze, for example, medieval French, because the syntax is different from the general syntax of modern French, and including it in our grammar would cause the grammar to wildly overgenerate analyses. But we make the same choice about classified advertisements (`particulier vend futon`), proverbs (`Pierre qui roule n'amasse pas mousse`), recipes (`Stir without stopping`), etc. These choices are task and corpus specific. For our corpus, for example, we allow NP as well as S to be generated from the start symbol of our grammar, in order to handle technical terms that appear as section headers.

Scoping ambiguities such as PP attachment are notoriously difficult. They have often been dealt wiih by ranking the different analyses, using either statistics or linguistic intuition. There

[2]There is another possible type of tagger error: a tagger could systematically mis-tag a word so that across the entire corpus it is never tagged with a POS that it should be tagged with. In practice this has not been a problem.

[3]The actual choice is between having one analysis for both of them, which will be correct for one and incorrect for the other, or having both a correct and an incorrect analysis for each of them.

is a vast literature on PP attachment, and ranking heuristics can be drawn from it.[4] LFG provides no formal mechanism for preferring one syntactic analysis over another. This may be a shortcoming if LFG is to be used for language engineering, where producing the n best analyses is commonplace.

2.3.3 Coping with Multi-Word Expressions

Multi-word expressions are quite varied in their characteristics. Here we want to focus on MWEs that have idiosyncratic syntactic behavior. The simplest case is an MWE that should always and only be analyzed as a single word. An example is `a priori`. This type of MWE can be handled in the tokenizer, so that `a priori` is a single lexical item that happens to contain a space. In English, `out of` is an MWE that functions as a preposition. We would not want to add a rule `PP` $\rightarrow$ `P P` to our grammar to handle this MWE, but we do need to analyze sentences in which it occurs. It is less clear that we can always treat it as a single word: is the `out` in `He came out of the house` part of the "word" `out of`, or a particle that goes with `came`? Other cases are clear. We mentioned previously `bien que` in French, which may be both an MWE and a sequence of two words. It is simply wrong to always consider it as one item.

A nondeterministic tokenizer can return both the two-token sequence `bien` followed by `que` and a single token `bien que`, where the lexicon contains all three, crucially with the last listed as a conjunction. It is left to the grammar to sort out which tokenization succeeds. Such a non-deterministic tokenizer has been built for French MWEs [CT96] and could be used as another module within the general parsing architecture. This mechanism does not in itself reduce ambiguity[5] but merely recognizes existing ambiguity. However, moving this recognition to the tokenizer simplifies the grammar, and may in fact reduce spurious ambiguity, because if rules are needed to glue elements such as `bien` and `que` together and give the result the right category, these rules may apply too widely.

A tokenizer can recognize only fixed expressions. There are variable MWEs that cause quite a bit of difficulty in parsing. An example of this class of MWEs is time expressions. An expression such as `Wednesday, August 28th` could be analyzed compositionally as an NP (a noun compound), but as a time expression, it also functions as an adverbial.

Having the grammar admit NPs as adverbials would vastly increase the number of analyses. The alternative of encoding a grammar for time expressions within the general grammar adds to the complexity (and the brittleness) of the grammar.

While the tokenizer cannot recognize these phrases because of their variation, they are still regular. A finite state transducer (FST) can transduce an MWE into a single token and give it the desired category, regardless of the category that a compositional analysis would impose upon it. An FST comprising a time-expression grammar could be interposed between the tokenizer and the parser. Such a transducer could output a single "word" of category, say `Date`; that is, it could produce from the MWE `Wednesday, August 28th` the token `#DATE#`, which could play the role of either an NP or an AdvP. The semantics—which date it is—need not be lost. In our system, the transducer would probably produce something like the output of the morphological analyzer, with the semantics encoded in tags: `#DATE# -Date -Wednesday -August -28th`. Grammar rules would then instantiate these tags as values of the f-structure for `#DATE#`.

Such FSTs, known as "local grammars", have not yet been incorporated into our LFG parsing system, but have been successfully used in other contexts in our group.

[4] We have not seen, in this literature, a discussion of preferring arguments to adjuncts (via subcategorization frames), which strikes us a valid general preference. Of course, in many cases the question of whether to consider something an argument or an adjunct is no more solved than PP attachment, so this will actually help only in clear cases.

[5] A choice could be made based on frequency of occurrence in a corpus, if desired, which would reduce ambiguity at the expense of occasional inaccuracy.

3 Conclusion

The techniques for eliminating analyses described in Section 2.2 are general. Here we want to note that reasons for rejecting an analysis may have little to do with correctness as defined by any linguistic theory: the domain under consideration may use a specialized lexicon; an analysis may be wrong due to lexical semantics that cannot currently be modelled in the theory; and as noted above, the number of analyses may be crucial. Grammar engineers think twice about changing the grammar to allow a second, equally viable, analysis of some constituent—and then don't do it—because it may double the number of results.

The challenge of the grammar engineer is to constrain the ways in which ambiguities combine. This challenge carries all of the constraints that are mentioned in Table 1 for an NLP grammar, but as previously mentioned, the most severe is the requirement that few analyses be returned. This requirement is relatively easily met in toy grammars, but toy grammars are insufficient for language engineering. If industrial-strength grammars are to be used in NLP, it is necessary to deal head on with the problem of ambiguity.

Although it is to be hoped that future progress in formal semantics, lexical semantics, pragmatics, and their interaction with syntax, will provide well motivated and efficient methods for reducing the set of syntactic analyses of a sentence, we are interested in language engineering today. Grammar engineers and computational linguists should investigate techniques, tools, and heuristics that can be used to eliminate ambiguity at various stages of processing. In this paper we have described several techniques for reducing lexical and syntactic ambiguity, including techniques for handling multi-word expressions.

References

[Av96] Andrews, Avery, 1996. *Avery Andrews' LFG system.* Software. Information at http://www.anu.edu.au/linguistics/software/lfgpc.html and http://online.anu.edu.au/linguistics/software/lfgpc.html.

[DKMZ95] Dalrymple, Mary and Ronald M. Kaplan, John T. Maxwell III, Annie Zaenen (eds.). 1995. *Formal Issues in Lexical-Functional Grammar.* CSLI Lecture Notes No. 47. CSLI Publications, Stanford, CA.

[Eu93] Bennett Paul and Paggio Patricia (eds.). 1993. *Preference in Eurotra.* Studies in machine translation and natural language processing, volume 3. Commission of the European Communities.

[Br82] Bresnan, Joan (ed.). 1982. *The mental representation of grammatical relations.* Cambridge, MA: The MIT Press.

[BT96] Boitet, Christian and Mutsuko Tomokiyo. 1996. *Theory and Practice of Ambiguity Labelling with a View to Interactive Disambiguation in Text and Speech MT.* in Proceedings of the 16th International Conference on Computational Linguistics (COLING-96):119-124.

[Ch93] Chanod, Jean-Pierre. 1993. *Problmes de robustesse en analyse syntaxique.* Actes de la confrence Informatique et langue naturelle, IRIN, Universit de Nantes, dcembre 1993.

[CT96] Chanod, Jean-Pierre and Tapanainen Pasi. 1996. *A Non-Deterministic Tokeniser for Finite-State Parsing.* ECAI '96 workshop on Extended Finite State Models of Language, Budapest, 1996.

[KM93] Kaplan, Ronald M. and John T. Maxwell. 1993. *LFG grammar writer's workbench.* Technical report, Xerox PARC. Available at ftp://ftp.parc.xerox.com/pub/lfg/lfgmanual.ps.

[Ko96] Kohl, Dieter. 1996. *The Charon System.* Software. Available at ftp.ims.uni-stuttgart.de:/pub/Charon.

[Hi94] Hindle, Donald. 1994. *A Parser for Text Corpora* In *Computational Approaches to the Lexicon*, Atkins, B.T. S, and Zampolli, A, eds. Oxford University Press.

[Ke95] Keogh, Ester. 1995. *Handbuch der Konstanzer LFG-Umgebung.* Working Paper No. 70. Sekretariat der FG Sprachwissenschaft, Universitaet Konstanz, Postfach 556O D185, D-78434 Konstanz, Germany.

[Ma96a] Mayo, Bruce. 1996. The Konstanz LFG Workbench. Software. Contact Bruce Mayo (bruce.mayo@pan.rz.uni-konstanz.de).

[Ma96b] Mayo, Bruce. 1996. *Die Konstanzer LFG-Werkbank*. To appear, LDV-Forum.

[Mo96] Montemagni Simonetta, Federici Stefano, Pirrelli Vito. 1996. *Resolving syntactic ambiguities with lexico-semantic patterns.* Proceedings of COLING 96, Copenhagen, Denmark, pp. 376-381.

[Va96] Vanderwende Lucy. 1990 *Using an on-line dictionary to disambiguate verbal phrase attachment.* Proceedings of the 2nd IBM conference on NLP, Paris, France, pp. 347-359.