Ambiguity Procrastination

Elaine Rich
Jim Barnett

Kent Wittenburg
David Wroblewski

MCC
3500 West Balcones Center Drive
Austin, Texas 78759

Abstract

In this paper we present the *procrastination* approach to the treatment of ambiguity, particularly in the context of natural language interfaces. In this approach, we try not to notice ambiguity unless we have the knowledge to resolve it available. In order to support this approach, we have developed a collection of structures that describe sets of possible interpretations efficiently. We have implemented this approach using these structures in Lucy, an English interface program for knowledge-based systems.

I. Introduction

In this paper, we will describe our work on Lucy, a system whose specific goal is to be a portable English front end subsystem for knowledge-based, interactive computer systems. The entire design of this system has been influenced by the decision to procrastinate resolving all kinds of ambiguity for as long as possible. The ambiguity procrastination approach is motivated by two concerns:

- The desire to minimize search by avoiding branching whenever possible.
- The desire to simplify the semantics of the processing routines by making them monotonic. By this we mean that we try to avoid making assertions that may need to be changed later.

Although Lucy is designed to be a portable system, it is currently implemented as a front end to a help system for a statistics program with an icon-based interface. Processing in Lucy is divided conceptually into the following parts: morphological analysis, syntactic analysis, semantic analysis, and discourse processing.[1] It happens that these parts occur in this order as well, although there is no real commitment to that in the design of the system, and we intend to explore more flexible control structures. Interestingly, many of the problems that have traditionally befallen such lock-step language processing systems become less serious when the philosophy of ambiguity procrastination is followed carefully. Ambiguity procrastination forces a novel treatment of most components of the language processing task. In particular, it forces a clear articulation, for each such component, of exactly what information that component contributes to the final interpretation. In the rest of this paper we will describe through examples the way that we have structured the main components of Lucy in order to support the notion that decisions should be made only when justified or necessary.

[1] In this paper we will focus primarily on syntactic and semantic analysis, because they are the best-developed parts of the system.

II. Noun Compounds

The first sentence we will consider is, "I compute the mean price data." Following morphological processing, which in this example is trivial, the sentence is parsed. The output of Lucy's parser is a description of the main constituents of the sentence. It is not a complete structural description since such a description cannot be built using only syntactic knowledge. Rather than trying to exploit other modules that contain the required nonsyntactic knowledge (as is done, for example, in other systems such as [Woods 80]), Lucy's parser simply does not attempt to form a complete structural description.

A simplified version of the result of parsing this sentence in Lucy is shown in Figure 1. Lucy's parser [Wittenburg 86a] is a best-first chart parser built on the unification formalism of [Shieber 84]. The grammatical framework it uses is a form of combinatory categorial grammar [Steedman 85, Wittenburg 86b]. For this example, the parser determines that there is a verb *compute*, whose subject is *I* and whose direct object is the noun compound *the mean price data*. The parser does not attempt to determine "case role" assignment. Nor does it attempt to assign an internal structure to the noun compound. Instead, it represents the compound with the noncommittal structure of modifiers (`mods`) and domains of modification (`doms`):

```
[mod: [lex: mean]
 dom: [mod: [lex: price]
       dom: [lex: data]]]
```

This structure is interpreted by later parts of Lucy as representing the entire *family* of parses that would be found if all bracketings were enumerated. To implement this approach, the grammar must be designed to guarantee that only this structure can be built. If the low attachment structure were built, it would describe a different and smaller family of interpretations. Using the Lucy grammar, this structure cannot arise because nouns cannot combine directly. To form a noun compound, a unary rule (of the form $x \rightarrow y$) must convert the first noun into an adjective. This unary rule only applies to single-word constituents, which forces the right-branching structure shown above.

```
[cat:  s
 pred:
  [cat: vp
   main-verb: [lex:compute]
   compls:
    [obj: [cat: np
           spec: [lex: the]
           head:
            [cat: cn
             mod: [lex: mean]
             dom: [cat: cn
                   mod: [lex: price]
                   dom: [cat: cn
                         lex: data]]]]]]
 subj: [cat: np
        lex: I]]
```

The Simplified Output of the Parse

Figure 1: "I compute the mean price data."

The result of the parsing process is passed to semantic interpretation. The goal of this step is to map from a string of English words into a set of assertions that are stated in terms of the knowledge base used by the backend program to which we are providing an interface. Thus it is this step that provides the bridge between English and domain knowledge. The first step is to convert the graph produced during parsing into a logical form. This Initial Logical Form (or ILF) is similar to the ILF described in [Hobbs 85]. It contains the information from the parser output that may play a role in semantic interpretation.[2] Specifically, it enumerates the entities contained in the sentence[3] as well as the surface functional relationships among those entities. The production of ILF from the parse graph is straightforward and uses no additional knowledge (except for handling idioms, which we will discuss in the next example). The ILF for the sample sentence is shown in Figure 2.

The predicates in the ILF are still the English words from the sentence or they are special system predicates, which are prefixed with *. The first argument of every predicate is a referent that corresponds to the event or state being described in the clause from which that predicate was derived. In the example, this is E-155. Roughly, the important assertions in the ILF can be read as follows:

2-5: There is a noun compound labelled X-158 composed of three parts, mean, price and data.
6: The event of the sentence has been described by the verb *compute*. The subject (represented as the second argument of the assertion) of the verb is *I* (X-156) and the direct object of the verb is the compound noun phrase *mean price data* (X-158).

```
1       ((I E-155 X-156)
2        (*COMPOUND E-155 X-158
3          ((MEAN E-155 Y-162)
4           (PRICE  E-155 Y-159)
5           (DATA E-155 Y-160))
6        (COMPUTE E-155 X-156 X-158)
7        (THE E-155 X-158)
8        (*QUANT-LIST(E-155 X-156 X-158)))
```

The Initial Logical Form

Figure 2: "I compute the mean price data."

In this structure, it is significant that the object of the verb is the entire noun compound, not any one of its pieces. This matters because it can happen, and does in this example, that the referent of a compound noun phrase is not a thing of the type given by the head noun. The final interpretation of the phrase *mean price data* will turn out to be a mean whose input was price data.

The second step of semantic interpretation in Lucy uses the expressions in the ILF as the basis for constructing the Final Logical Form (FLF). The FLF contains a description of the meaning of the sentence in terms defined within the backend knowledge base. The FLF construction algorithm selects expressions one at a time from the ILF. When an expression is chosen, the system's action is determined by the predicate in the expression. If the predicate is an English word, then the predicate is looked up in the semantic lexicon, which is where the connection between English words and objects in the backend knowledge base is defined. Each entry consists of a set of constraints that a meaning of a word imposes on the interpretation of a sentence containing the word. For example, the entry for the predicate COMPUTE is

```
(compute x y z) →
   (isa x computational-process)
   (has-agent x y)
   (person y)
   (output x z)
   (computable-object z)
```

These constraints are added to the constraints that have been imposed on the interpretation of the sentence by the ILF forms that have already been processed.

If an English word is ambiguous with respect to the backend knowledge base, then there may be more than one set of constraints that could be added to the interpretation. This may result in a branch in the interpretation process if more than one set of constraints is internally consistent.[4] When search is required, a best-first search procedure is used, where best is defined by priorities such as those given to lexical entries and to common attachment structures. In this example, the word *mean* is ambiguous. It has two

[2]There is, in addition, a separate syntax structure that provides additional information that is necessary for anaphora resolution, but we will ignore that here.

[3]This list of entities includes a set of discourse referents in the sense of [Kamp 81] as well as a set of entities that do not have that status but that will eventually correspond to knowledge-base concepts.

[4]It is because of the possibility of branching that this algorithm is not purely a constraint satisfaction process such as that described in [Mellish 85]. It is, however, an instance of the generalized constraint satisfaction mothod described in [Rich 88]. In addition, we are exploring ways of decreasing branching at this point through the use of a richer constraint language.

```
(ASSERT NIL
(E-155 (ISA COMPUTATIONAL-PROCESS)
       (HAS-AGENT X-156)
       (OUTPUT X-158))
(X-158 (ISA MEAN) (SPEC DEF) (INPUT Y-160))
(Y-160 (ISA DATASET)
       (NAMED-BY-STRING X-159))
(X-159 (ISA STRING-CONSTANT) (NAME PRICE))
(X-156 (ISA USER) (NAME CURRENT-USER)))
```

The Final Logical Form

Figure 3: "I compute the mean price data."

interpretations with respect to the backend system: a concept (`mean`) corresponding to the mathematical notion of a mean, and a concept (`mean-calculation`) corresponding to the particular function that computes the mean in the system being discussed. Since only the former can be computed, the correct interpretation for the word *mean* can be found as soon as the constraints imposed by the word *compute* are posted. Examples such as this point up the importance of developing powerful mechanisms for handling ambiguity when the meaning of English sentences must be defined in terms of a detailed knowledge base that contains highly differentiated, specialized concepts with respect to which English words are usually highly overloaded.

If the predicate is a special system predicate, then a system-specified action occurs. In this example, `*COMPOUND` does occur and it is here that a decision about both the structural associativity and the semantic connections among the words in the noun compound must be resolved. The ambiguity that was procrastinated during parsing can now be dealt with by exploiting semantic knowledge and the mechanisms of constraint satisfaction. To determine the structure of the noun compound *mean price data*, Lucy does the following. Starting at the right (where the most likely head noun is), it looks up each noun in the semantic lexicon to find its meaning(s) (stated as set(s) of constraints). It then looks to see if there is any information on how the concept represented by the word combines semantically with the concepts represented by the other words. This information must be stored with individual concepts because there is no general rule for computing such semantic relations (as shown by the classic example: *olive oil, corn oil, peanut oil, baby oil*). This is particularly true given the necessity to map correctly into semantic relationships appropriate to some externally specified knowledge base.[5]

The FLF for the entire sentence is simply the union of the individual sets of constraints, sorted by the objects to which they apply, and with more general assertions eliminated if more specific ones are present. The FLF for our first example sentence is shown in Figure 3.

Actually, the complete FLF must also include the results of discourse processing, which, in this example, means finding the referent for the definite noun phrase *the mean price data*. This can be done by using the FLF shown in the figure as input to the procedure that finds objects in the current discourse context that match the constraints given in the noun phrase.

[5]This approach contrasts in spirit with approaches such as [Isabelle 84], in which an attempt is made to find a more principled basis for semantic composition.

III. Idioms and Postmodification

The second sentence we will consider is, "The system looks up the word for the user." A simplified form of Lucy's parse of the sentence is shown in Figure 4*a*. There are two structural ambiguities in this sentence:

- The attachment of the prepositional phrase *for the user*.
- The choice between *look up* as a two-word verb with a direct object, versus *look* as an intransitive verb followed by the prepositonal phrase *up the word*.

Since decisions on these issues cannot be made using purely syntactic information, they are not made during parsing in Lucy. Postmodifier structures (such as prepositional phrases and relative clauses) are represented using a modifier/domain structure of the form:

```
[MOD:  <postmodifier phrase>
 DOM:  <tree of structure within
        which MOD attaches>]
```

This structure will be interpreted during ILF construction as allowing the modifier to attach anywhere on the rightmost branch of the structure given in DOM.[6] We guarantee that this is the only structure that the parser can build by blocking the construction of any modifier that is itself modified.

Two-word verbs are handled by designing a grammar that guarantees that only a single interpetation can be produced. The word *up* will be treated as a preposition and not as a particle except when the particle interpretation is unambiguous (as it would be in *look the word up*). The parse result containing the prepositional phrase *up the word* will not be interpreted during ILF as actually committing to that interpretation, however.[7]

[6]Actually, the situation may be a bit more complicated in the case of extraposition [Wittenburg 87].

[7]The hypothesis lurking behind this design is that it may be possible for a grammar used only for a first pass in processing to be unambiguous in its assignment of initial bracketings to strings. Subsequent processes that make use of semantic (and possibly discourse) information would then branch in assigning interpretations to the initial bracketing. While basic attachment ambiguities are relatively amenable to such a treatment, we have found the interactions to be complicated in other cases. In the case at hand, for instance, there are interactions in the grammar between the various complement structures possible for *look*, the ambiguity between *up* as a preposition or a particle, and the attachment of the preposition. It is difficult, if not impossible, to maintain monotonicity in the system when interactions are this complex and at the same time maintain traditional assumptions about syntactic bracketing. The working version of Lucy attempts to maintain an unambiguous grammar at the cost of a rather strained relation between syntactic bracketings and semantic interpretations. Forthcoming work will discuss the pros and cons of such an approach.

Construction of the ILF for this sentence requires the use of the idiom lexicon to recognize the idiom *look up*. The recognition of possible idioms at this point in processing is, to some extent, a violation of the ambiguity procrastination principle, because the possibility of idioms is detected (resulting in a branch in the processing) before the information that is required to choose between the idiomatic and nonidiomatic readings is available (since no other semantic information is available until FLF construction begins). The reason for this violation is that it permits a significant structural commitment to the FLF construction process to be made. In Lucy, construction of the FLF from the ILF is completely compositional. Each ILF form adds constraints to the FLF but never modifies the constraints imposed by any other ILF form. Unfortunately, the words in idioms cannot be processed that way. *Kicking the bucket* does not involve any kicking or any buckets. In essence, the two motivations behind the procrastination principle are in conflict here. By making the compromise of inserting idioms during ILF construction, we make it possible to use a monotonic constraint posting algorithm in building the FLF. Because constraint posting based on local information is itself such a powerful way of handling ambiguous structures, the overall goal of ambiguity procrastination is best served by introducing idiom-induced ambiguities at the beginning of semantic processing.

For this example, the ILF is shown in Figure 4*(b)*. The ***OR** represents the choice between the idiomatic (**DICT-SEARCH**) and the literal (**LOOK**) meanings of the verb phrase. Notice that the choice of an attachment point for the prepositional phrase *for the user* has still not been made. Instead, each of the verb phrase alternatives contains an ***ATT**achment list, containing, in the order in which they occurred in the sentence, attachment points and attachable structures. In the case of the nonidiomatic reading, ***ATT** contains the referent corresponding to the entire event and a description of each of the attachable things in the sentence (namely the two prepostional phrases). This list will be interpreted as allowing the *up* phrase to attach only to the event and as allowing the *for* phrase to attach either to the event or to *word*. Given the idiomatic reading (in which *look up* is a two word verb), there is only one attachable thing, which corresponds to the prepositional phrase *for the user*. But ***ATT** also contains the referent corresponding to the entire event and the referent corresponding to *word*, since both of these are possible attachment points for the final prepositional phrase (*for the user*). Because the attachments of the prepositional phrases have not yet been determined, it is not possible to specify both the arguments of the relations represented by the prepositions. The dummy argument **ARG1** is used in the ILF to represent the unknown argument. It will be bound when an attachment point for the phrase has been selected.

Just as in the previous example, the predicates in the ILF still correspond to the English words in the surface string. This means that some of the predicates carry little information on their own. In this sentence, an example of such a "vague predicate" [Martin 83] is **FOR**, which may end up specifying any of a large number of concrete relationships between its arguments. This choice, though, must depend on the arguments themselves.

```
[cat: s
 mod: [cat: pp
       prep: [lex: for]
       pobj: [cat: np
              spec: [lex: the]
              head: [cat: cn
                     lex: user]]]
 dom: [cat: s
       mod: [cat: pp
             prep: [lex: up]
             pobj: [cat: np
                    spec: [lex: the]
                    head: [cat: cn
                           lex: word]]]
       dom: [cat: s
             pred: [lex: look]
             subj: [cat: np
                    spec:[lex:the]
                    head:[lex:system]]]]]
```

(a) The Simplified Output of the Parse

```
((SYSTEM E-171 X-176)
 (WORD E-171 X-174)
 (USER E-171 X-172)
 (*OR ((LOOK E-171 X-176)
       (*ATT (E-171)
             (UP E-171 ARG1 X-174)
             (FOR E-171 ARG1 X-172))
    ((DICT-SEARCH E-171 X-176 X-174)
       (*ATT (E-171) X-174
             (FOR E-171 ARG1 X-172)))))
 (THE E-171 X-176) (THE E-171 X-174)
 (THE E-171 X-172))
 (*QUANT-LIST (E-171 X-176 X-174 X-172))
```

(b) The Initial Logical Form

```
(ASSERT NIL
 (E-171(ISA DICT-SEARCH-FOR)(AGENT X-176)
       (BENEFICIARY X-172)(OBJECT X-174))
 (X-172(ISA USER) (SPEC DEF))
 (X-176(ISA PRGRM)(SPEC DEF)(NAME VSTAT))
 (X-174(ISA LEX-ITEM) (SPEC DEF)))
```

(c) The Final Logical Form

Figure 4: "The system looks up the word for the user."

The FLF (shown in Figure 4*(c)*) is built from the ILF using the best-first constraint posting procedure described for the first example sentence. In the current implementation, branching occurs as a result both of the ***OR** and of the ***ATT**. However, there may be ways of avoiding at least the latter of these through the use of an appropriate constraint language.

IV. Quantifier Scope

The third sentence we will consider is, "Every command computes a function." The interesting ambiguity in this sentence is the scope of the quantifier *every*. The FLF for this sentence is shown in Figure 5. It represents an assignment of wide scope to the quantifier *every*, with the result that the referent of *a function* must depend on a particular value of the referent of *command*. This is indicated by the assertion `(F X-146)` that is made about the object `X-148`. This assertion states that the referent of `X-148` depends on (is a function of) the referent of `X-146`. This reading was simply chosen as the default reading. This mechanism for choosing a representation is not very interesting. But the simple mechanism for representing the choice is interesting, among other reasons, because it can also be used to represent the fact that no choice has yet been made. In this case, no `F` assertions are specified.

```
(EVERY X-151 AX-153)

(ASSERT NIL
 (E-145 (ISA COMPUTATIONAL-PROCESS)
        (HAS-INSTRUMENT X-146)
        (OUTPUT X-148))
 (X-148 (ISA FUNCTION) (F X-146))
 (X-146 (ISA USER-OPERATOR) (QUANT ALL)))
```

The Final Logical Form

Figure 5: "Every command computes a function."

Representing an incomplete assignment of quantifier scope has been a problem for other natural language understanding systems [Hobbs 83, Woods 77]. The approach used in Lucy is an attempt to satisfy the major requirements for an acceptable representation, namely that it should support reasoning, that it should be easily modifiable as new information is obtained, and that it should not impose levels of complexity on the overall representation that are not otherwise necessary. The idea is similar to the idea of representing scope-induced dependencies as Skolem functions and it shares with that technique the property that knowledge about scope relationships is contained in separate assertions that can be made as appropriate, rather than being built into the structure of the overall content of the sentence. But our technique, which is essentially an implementation of the representation developed in Discourse Representation Theory [Kamp 81], avoids the pitfalls of a Skolem function approach, since it does not require an initial commitment on the question of which objects are universally quantified and which are existentially quantified. Instead, existential quantification will arise whenever no other quantification applies. Thus a late commitment on this question too is possible. This explains the lack of an explicit quantificational statement in the FLF corresponding to the noun phrase *a function*.

V. Conclusion

The goal of this paper has been to show how a commitment to the principle of ambiguity procrastination can shape the design of a natural language understanding system. Implementing this commitment requires a clear articulation of the contribution of each part of the understanding system. It also requires the development of a family of representational and processing techniques that support the manipulation of incomplete structures. Some of the techniques that Lucy uses to do this have been derived from other work in this area. See, for example, [Church 80], [Church 82], [Marcus 83], [Pereira 83] for discussions of ways to represent specific kinds of syntactic ambiguity. What we have tried to do in the design of Lucy is to build on these techniques in a unified way to reduce the overall complexity of the language understanding process.

What we have actually succeeded in doing is to procrastinate several kinds of ambiguity through syntactic processing so that they do not show up until semantic processing time, when the knowledge to deal with them is available. Unfortunately, in the current system, it is often the case that branching does then occur. Two comments are worth making on this, since one might ask the question, "What have you gained if you eventually branch anyway?"

The first response is that we had a strong motivation to avoid the production of multiple parses. Syntactic parsing operates in a different search space than do all the other understanding processes. It operates in a space of (partial) structural descriptions. The other processes (including the ILF to FLF process described in this paper, as well as other processes such as anaphora resolution and pragmatic inference) operate in a space of sets of logical assertions. It is much easier to conduct a search in a single space than it is to move from one space to another. Thus there is an advantage to pushing ambiguity forward so that all of it is handled in a single search space.

The second response is that it is possible (although not yet implemented in Lucy) to reduce the branching in the space of logical assertions through the use of one more general (i.e., weaker) assertion rather than a set of alternative stronger ones. For example, rather than listing the alternative exact interpretations of the word *mean*, we could have stated only that some kind of averaging is involved, leaving the addition of the specific facts (namely the choice between a mathematical concept and a VSTAT function) to be added by some other part of the sentence (such as the verb *compute*). If the information required to choose among the detailed meanings must come from some other part of the sentence anyway, then it is unnecessary to add that information twice.[8] If the entire search process within the logical assertion space is viewed as one of constraint satisfaction, then this follows naturally.

[8] Another way of saying this is that, in a constraint satisfaction system, every inconsistency that arises during the solution of an initially consistent problem corresponds to a situation in which some module made an unjustified commitment to something. A good way to improve the performance of such a system is to eliminate such early commitments.

These two responses taken together suggest that if syntactic processing can be unambiguous and if the right constraints can be articulated for semantic and pragmatic processing, then the total branching level may be able to be reduced. Of course, an alternative that produces the same result would be to allow ambiguity to be detected during syntactic processing but to redescribe syntactic processing as yet another source of constraints that can be applied within the same space as is other processing. We are investigating both these approaches.

References

[Church 80] Church, K. On Memory Limitations in Natural Language Processing. Master's thesis, MIT, 1980.

[Church 82] Church, K., and R. Patil. Coping with Syntactic Ambiguity or How to Put the Block in the Box on the Table. *Journal of Computational Linguistics* 8:139-149, 1982.

[Hobbs 83] Hobbs, J. R. An Improper Treatment of Quantification in Ordinary English. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages 57-63. 1983.

[Hobbs 85] Hobbs, J. R. Ontological Promiscuity. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*. 1985.

[Isabelle 84] Isabelle, P. Another Look at Nominal Compounds. In *Proceedings of Coling84*. 1984.

[Kamp 81] Kamp, H. A Theory of Truth and Semantic Representation. In J. Froenendijk, T. Janssen, & M. Stokhof (editors), *Formal Methods in the Study of Language, Part 1*. Mathematisch Centrum, Amsterdam, The Netherlands, 1981.

[Marcus 83] Marcus, M. P., D. Hindle, & M. M. Fleck. D-theory: Talking about Talking about Trees. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*. 1983.

[Martin 83] Martin, P., D. Appelt, & F. Pereira. Transportability and Generality in a Natural-Language Interface System. In *Proceedings IJCAI 83*. 1983.

[Mellish 85] Mellish, C.S. *Computer Interpretation of Natural Language Descriptions*. Halsted Press, New York, 1985.

[Pereira 83] Pereira, F. *Logic for Natural Language Analysis*. Technical Report, SRI International, 1983.

[Rich 88] Rich, E. *Artificial Intelligence, Second Edition*. McGraw-Hill, New York, 1988.

[Shieber 84] Shieber, S. The Design of a Computer Language for Linguistic Information. In *Proceedings of COLING-84*. 1984.

[Steedman 85] Steedman, M. Dependency and Coordination in the Grammar of Dutch and English. *Language* 61:523-568, 1985.

[Wittenburg 86a] Wittenburg, K. A Parser for Portable NL Interfaces Using Graph-Unification-Based Grammars. In *Proceedings AAAI 86*. 1986. Also available as Technical Report MCC HI-179-86.

[Wittenburg 86b] Wittenburg, K. *Natural Language Parsing with Combinatory Categorial Grammar in a Graph Unification-Based Formalism*. PhD thesis, Department of Linguistics, The University of Texas at Austin, 1986.

[Wittenburg 87] Wittenburg, K. Extraposition from NP as Anaphora. In *Syntax and Semantics, Volume 20: Discontinuous Constituencies*. Academic Press, New York, 1987. Also available as Technical Report MCC HI-118-85.

[Woods 77] Woods, W. Semantics and Quantification in Natural Language Question Answering. In *Advances in Computers Volume 17*, pages 1-87. Academic Press, New York, 1977.

[Woods 80] Woods, W. A. Cascaded ATN Grammars. *American Journal of Computational Linguistics* 6(1):1-12, 1980.