ULINK: A Semantics-Driven Approach to Understanding Ungrammatical Input

Jeffrey D. Kirtner
Steven L. Lytinen
Artificial Intelligence Laboratory
The University of Michigan
Ann Arbor, MI 48109
e-mail: jeff@caen.engin.umich.edu

Abstract

This paper describes ULINK, a program designed to understand ungrammatical input. While most previous work in the field has relied on syntactic techniques or sublanguage analysis to parse grammatical errors, ULINK uses a semantics-driven algorithm to process such input. The paper gives a brief overview of LINK, the unification-based system upon which ULINK is built; special attention is given to those aspects of LINK which allow ULINK to use semantics to process ill-formed input. The details of ULINK's algorithm are then discussed by considering two examples. The paper concludes with a discussion of related research and problems which remain to be solved.

Introduction

Traditional Natural Language Processing (NLP) systems have a difficult time understanding ungrammatical sentences. These systems have separate modules to analyze the syntax and semantics of a sentence (Winograd 1972, Hirst 1983, Shieber 1986). Typically they first parse a sentence (or parts of a sentence) based on syntax and then present the semantic module with the various possible parses. The semantic module then determines which of the possible parses makes sense, given its knowledge of the words in the sentence and the syntactic constituents determined by the parser.

Ungrammatical input causes two problems for traditional systems. First, the syntax module usually contains the 'proper' syntax of a language, and hence is unable to parse ungrammatical input in the first place. Second, for some kinds of text, such as terse text in which many words are left out of a sentence, there are many possible syntactic interpretations of a sentence, only a few of which make semantic sense. Even if the parser overcomes the first problem and can parse ungrammatical input, the second problem leads to inefficient analysis, since the system must consider many interpretations of each sentence.

A system called ULINK was designed in an attempt to overcome these two problems. ULINK's grammar contains only the proper syntax of English, but in the face of grammatical errors ULINK attempts to relax the syntactic constraints in ways which make semantic sense. At the same time, ULINK uses semantic information to consider only semantically reasonable parses of the sentence, leading to efficient parsing of terse text.

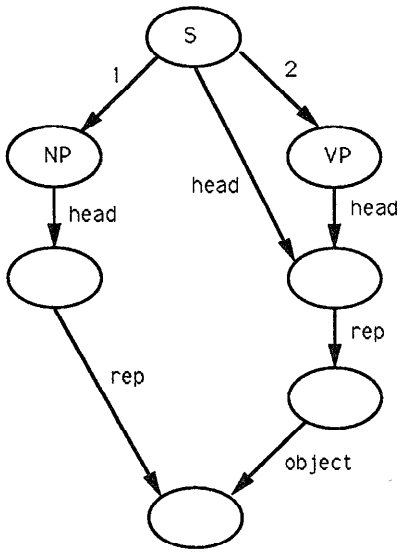
ULINK is based on LINK, an integrated unification-based NLP system developed by Steven Lytinen (Lytinen and Roberts 1989, Lytinen 1990). ULINK's domain is a set of automobile stalling cases taken from a database of car problems. The cases are one-line problem descriptions which were entered into the database by mechanics in terse, ill-formed statements.

LINK itself cannot parse ungrammatical input because, like traditional systems, it is syntax-driven and cannot parse any sentence not acceptable to its grammar. Unlike traditional systems, however, syntactic and semantic processing in LINK are integrated in one module, thus making semantic information available during parsing. ULINK uses this semantic information to recover from grammatical errors and continue the parse.

As we will see later on, ULINK must perform two functions in order to parse ungrammatical input. It must be able to find semantic connections between the sub-constituents already built by the parser at the time that an error is detected; and it must be able to locate grammar rules based on the semantic connections the rules make. If ULINK can find both a semantic connection between two sub-constituents and a grammar rule to make that connection, then ULINK can apply the rule *as if* the correct syntactic constituents were present. In order to describe ULINK in more detail, a few more words must be said about LINK.

LINK

LINK encodes all syntactic, semantic, and pragmatic knowledge in unification constraint rules. Knowledge is given to the system in one of three ways: as word definitions in the lexicon, as grammar rules, and as pragmatic definitions of concepts in a semantic net. In all three cases, the knowledge is represented as a




```
S:
   (1) = NP
   (2) = VP
   (head) = (2 head)
   (head rep object) = (1 head rep)
```

Figure 1: Grammar rule and its associated DAG

Directed Acyclic Graph (DAG).

Consider the S rule in Figure 1. The rule specifies a set of constraints which any node labelled S must have. The constraints consist of a *path*, or a sequence of arcs with the appropriate labels starting from the node in question; and a *value*, which is another node to be found at the end of the path. The values of constraints specify either the label of the node found at the end of the path, as in equations 1 and 2; or a *unification* with another path, as in equations 3 and 4.

The S rule encodes the following information about sentences. Equations 1 and 2 specify that sentences are made up of a noun phrase (NP) and a verb phrase (VP). Equation 3 specifies that the HEAD path of the sentence is to be unified with the HEAD path of the VP. It is via HEAD links that information gathered at lower levels of the parse is propagated up to higher levels; after unifying the two HEAD paths, any information that the VP has gathered will now be accessible to the S node. Equation 4 maps a syntactic constituent to its semantic role. In this case, the semantic representation of the NP becomes the object of the sentence. It is always the case that semantic information is stored under the (HEAD REP) path of a DAG.

Grammar rules in LINK are indexed by their subconstituents. Thus the S rule will be indexed under (NP VP), and the rule will only be accessible to LINK once the NP and VP constituents have been built.

In order to understand the changes ULINK makes to LINK to allow it to parse ungrammatical input, it will help to work through a sample parse. The sentence we will parse is 'The engine stalls.' Before working through the example we need to define the grammar rules for NP's, VP's, and determiners, as well as the words in the sentence.

```
S:
  ((1) = NP
   (2) = VP
   (head) = (2 head)
   (head rep object) = (1 head rep))
NP:
  ((1)             = DET
   (2)             = N
   (head)          = (2 head)
   (head common)   = (common))
VP: ((1)           = V
     (head)        = (1 head))

Lexical entries:
The: (DET (head rep ref) = definite)
Stalls: (V (head rep) = *stall*)
Engine: (N (head rep) = *engine*
           (head common) = common)

Pragmatic information:
(DEFINE-SC engine-action
  is-a (action stall-condition)
  formulae (((OBJECT) = *engine*)))
(DEFINE-SC *stall*
  is-a (engine-action))
```

The notation is mostly straightforward. '*stall*' and '*engine*' refer to the meanings of the words 'stall' and 'engine,' respectively; in general '*thing*' refers to the meaning of 'thing.' Pragmatic knowledge is input declaratively and is built into a semantic net. In this example a *stall* is defined to be an ENGINE-ACTION, and under ENGINE-ACTIONs we declare that the object of an ENGINE-ACTION is a *engine*. Pragmatic information is inserted into the DAG at the time a word is first read by the parser.

When parsing 'The engine stalls,' the parser first reads the word 'The' and builds a DAG labelled DET using information defined in the dictionary entry for the word 'The'. Next the parser reads the noun 'engine' and builds a DAG labelled N. The sub-constituents of the NP rule have now been built, so the parser applies the NP rule to the DET and N DAGs and builds a DAG labelled NP.

The parser then reads the word 'stalls' and builds a DAG labelled V; the V DAG includes the semantic information defined in the dictionary and the pragmatic information defined in the semantic net. This means that when the V DAG is built, the (HEAD REP) path of the DAG contains two pieces of information: that the representation of the verb is *stall*, and that the object of the verb is *engine*. After the DAG labelled V is built, LINK builds a DAG labelled VP from the V DAG.

Once the VP DAG is built, the NP and VP DAGs are

used to index the S rule. The S rule will apply successfully. Equation 4 is of particular interest here. It stipulates that in order to build an S node, the object of the VP must unify with the NP. In this case, both the object of the VP and the NP have the label *engine*, so they unify and the constraint is satisfied. Since LINK's goal is to build an S DAG which spans all the words of the input, LINK stops parsing. The meaning of the sentence is stored under the (HEAD REP) path of the S DAG. It looks like this:

```
*stall*
  Object: *engine*
  ref: definite
```

ULINK, LINK and Ungrammatical Input

Let us next consider the problem of parsing ungrammatical input. We will again work through a short example, but this time LINK will fail to parse the input. After demonstrating why LINK fails, it will be easy to explain the extensions ULINK makes in order to parse the same input.

The input we will try to parse is 'engine stalls.' LINK first reads the word "engine" and builds a DAG labelled N with the definition from "engine". Since no determiner is present, LINK cannot build a larger constituent from the noun, and so continues on to the verb. As before, LINK builds a V node and then a VP node, again attaching the information that the object of the verb is *engine*. Now LINK can go no further: it has built N and VP DAGs, but no rule is indexed under these constituents. In particular, the S rule requires an NP label, and so cannot be indexed. Thus LINK fails to parse the sentence.

It is clear, however, that the semantic information needed to combine the two constituents is readily available. The N has a semantic representation (HEAD REP) = *engine*, and the VP knows that its (HEAD REP OBJECT) = *engine*. It is reasonable to assume that there may be a valid connection between the N and the object of the verb. The problem is that we have no way of finding a rule to make that connection, because the rules are indexed only by syntactic constituents.

ULINK makes two extensions to LINK to allow it to use semantic information to recover from grammatical errors. First, ULINK searches the semantic information available in the DAGs already built at the time an error is detected, trying to find a possible semantic connection between two constituents; and second, it cross-references the grammar rules according to the semantic connections the rule can make, so that if semantic connection between two constituents is found, we can find a rule to apply to make that connection.

Consider the 'engine stalls' example. When the S rule was entered into the grammar, ULINK cross-referenced the rule by the OBJECT slot, since the S rule uses the OBJECT slot to make a semantic connection. When the error is encountered during parsing, ULINK discovers that there is a semantic connection in the OBJECT slot between the noun 'engine' and the VP 'stalls.' That is, the noun dag has a (HEAD REP) equal to *engine* and the VP has a (HEAD REP OBJECT) equal to *engine*. Since the semantic connection occurs in the OBJECT slot, ULINK looks for rules that make a semantic connection with an OBJECT. As mentioned before, the S rule makes this connection. Thus ULINK finds both the semantic connection and the rule to make that connection, so the S rule is applied as if the NP DAG had already been built.

The ULINK algorithm

The pseudo-code below describes the basic operations of ULINK. The three highlighted steps in ULINK's processing are described in more detail below. There are only two things to note here. The first is that ULINK's extensions are used only when syntax alone does not enable the parser to find any more grammar rules to apply; in this sense grammatical errors are thought of as exceptions. Second, whereas in LINK a parsing failure occurs when a grammatical error is detected, in ULINK a failure occurs when an error is detected *and* no more semantic connections can be found between the DAGs already built.

```
At system start-up:
<1> Cross-reference the rules by their
    syntactic constituents and by the
    semantic connections the rules make

Loop until success or failure:
 Try to find a rule through normal LINK
 processing;
  If rule is found Then
     Apply the rule;
  Else /* parse ungrammatical input */
     While (no new DAG has been built)
       <2> Find a semantic connection
           (independent of the grammar)
           between two adjacent DAGs;
           Find a rule to apply to make
           that connection;
       <3> If (constituents required by
           the rule are 'close enough'
           to the constituents of the
           DAGs) Then
             Apply the rule build a new
             DAG;
```

Cross-referencing the rules

In ULINK, grammar rules are indexed not only by syntactic constituents, but also via the semantic connections the rules make. ULINK makes the assumption that any rule with a constraint involving a (HEAD REP) path is a candidate rule for making semantic connections. This is reasonable, since the (HEAD REP) path stores the semantic representation of each constituent that is built. In the S rule above, for example, the fact that the constraint '(HEAD REP OBJECT) = (1 HEAD REP)' is in the rule qualifies the S rule to be cross-listed by OBJECT as well as by the traditional syntactic indexing based on the subconstituents (NP VP). As another example, the VP rule below would be cross-listed by the MODIFIES slot:

```
VP:
      ((1)          = VP
       (2)          = ADV
       (head)       = (1 head)
       (head rep)   = (2 head modifies))
```

Finding a connection between constituents

The algorithm to find a semantic connection begins by looking for a connection among DAGs covering the largest number of input words, and then continues searching shorter and shorter components until either a connection is found or no more DAGs remain to be searched. It only searches for connections between two adjacent DAGs. The algorithm examines the (HEAD REP) links of the two DAGs, since that is where the semantic information is stored. A connection is found whenever two DAGs share the same label (as in the example of engine and stall above), or whenever one DAG label is an instance of a class represented by the second DAG label (i.e. DAG1 IS-A DAG2).

Deciding whether or not to apply a rule

Suppose that a semantic connection between two DAGs has been found, and a rule has been found that makes the appropriate connection. It does not necessarily follow that the rule should be applied. There must be some determination that the syntactic constituents of the DAGs are "close enough" to the syntactic constituents required by the rule to warrant applying the rule to the DAGs. In the 'engine stalls' example, the constituents of the DAGs were N and VP, and the rule required an NP and VP. It is reasonable to assume that an N is close enough to an NP to apply the rule *as if* an NP had actually been built. Indeed, the algorithm actually changes the label of the DAG from N to NP before applying the rule. ULINK currently uses a global list of constituents considered to be close enough to other constituents to warrant applying rules to one constituent in place of the other.

A second example

Consider the sentence 'Engine stalls intermittent.' This sentence is ungrammatical because there is no determiner in the initial noun phrase, and because an adjective, 'intermittent,' is modifying the verb 'stalls.' Assume that we update our grammar rules and dictionary as follows. Figure 2 describes how the VP rule works by showing the results of unification under normal processing - that is, assuming the sub-constituents VP and ADV had been built.

```
Grammar rule:
VP:
  ((1)          = VP
   (2)          = ADV
   (head)       = (1 head)
   (head rep)   = (2 head modifies))
```

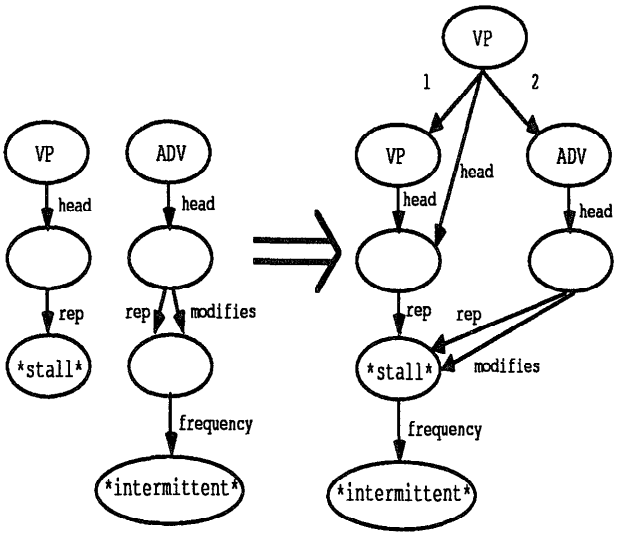


Figure 2: Unifying VP and ADV

```
Dictionary definition:
intermittent:
(ADJ
  (head rep) = ENGINE-ACTION
  (head modifies) = (head rep)
  (head rep frequency) = *intermittent*)
```

Let us follow ULINK's parse of the sentence 'Engine stalls intermittent.' At the time an error is detected, ULINK will have built both a VP DAG from the word 'stall' as described in the previous example, and a DAG labelled ADJ for the word 'intermittent.' The system will try to find a semantic connection between these two DAGs, and will find one because in the dictionary definition of the word 'intermittent' the unification constraint '(head rep) = ENGINE-ACTION' declares that 'intermittent' can modify an ENGINE-ACTION. Since a stall is an ENGINE-ACTION, as defined in the pragmatic knowledge given in the original example, then we have a connection between 'stall' and 'intermittent' (i.e. that a 'stall' is an ENGINE-ACTION and that 'intermittent' modifies ENGINE-ACTION's by declaring that their frequency is *intermittent*). This connection is made in the MODIFIES slot of the DAG labelled ADJ. Since ULINK has cross-referenced the grammar rule (VP ADV) by the MODIFIES slot, we can find a grammar rule to make the semantic connection we have found. Next ULINK determines that the ADJ label in the DAG is close enough to the ADV constraint in the rule to warrant applying the rule, as described in step three of the ULINK algorithm. Therefore the syntactic constraint requiring an ADV is relaxed, the rule is applied, a VP DAG is built, and normal processing continues. In this case, normal processing will result in the discovery of a second grammatical error; namely, that the parser can build an N node for 'engine' and a VP node for 'stalls intermittent,' but cannot combine an N and VP

into an S. The handling of this error was described in the first example of ULINK's processing, and the result here is the same. Thus the final result of the parse will be as follows:

```
*stall*
  Object: *engine*
  frequency: *intermittent*
  ref: definite
```

Discussion of Related Research

In the past there have been two general approaches to processing ill-formed text, one which relies on syntactic parsing techniques and the other which analyzes the input as a sublanguage. The syntactic techniques can be subdivided into two areas. The first area uses grammar-specific rules to recover from errors. Weischedel and Sondheimer called such rules Meta-rules; other systems using this approach include Jensen *et al* (1983). The second of the syntactic approaches uses grammar-independent rules that depend only on the grammar formalism used. Mellish 1989 sketches such a system based on an active chart parser. In some ways, Mellish's approach is similar to ours, in that explicit rules are not used to drive the process of matching an ungrammatical input to the system's grammar. However, ULINK uses semantic information to drive this matching process, whereas the approach of Mellish and others relies exclusively on syntactic features.

Encoding recovery rules as grammar-specific Meta-rules has two potential drawbacks, one concerned with coverage of errors and the other concerned with efficiency. It may turn out to be an extremely difficult task to encode every new ungrammatical construction into a meta-rule. Even if this can be done for a given database of sentences, it may be that the rules themselves are not transferable to another domain, which might contain a different set of ungrammatical constructions. ULINK escapes the transportability problem because its grammar contains only the correct grammar of the language, which is much less likely to change from one domain to another.

ULINK also has a possible processing advantage over both of the syntactic approaches mentioned above. Systems that rely on syntactic processing to recover from errors must apply any rule that fits the input. At least for corpora containing terse text, there are often many different syntactic constructions which fit the phrases built at the time an error is discovered. Usually only one or two of these constructions makes semantic sense. ULINK is able to use the semantics of the phrases built so far to point directly to the rule to use to recover from the error. Thus ULINK trades off a search for a semantic connection and an application of one grammar rule with no search for a connection but the application of many rules. Which of these approaches is more efficient is an empirical question that we will be addressing.

The second approach to ungrammaticallity is to treat the input as a sublanguage. This approach is described in Kittredge and Lehrberger (1982), Grishman and Kittredge (1986), and by work that has grown out of the Linguistic String Project at NYU (see for example Sager (1982); Marsh (1983); and Grishman *et al* (1986)). A sublanguage is characterized by a restricted domain and by greater syntactic and semantic regularities than are found in the language as a whole. A representative sample from the corpora can be statistically analyzed to find sublanguage word classes - sets of words that are acceptable in the same contexts. The word classes and the contexts in which they occur can then be explicitly encoded in the grammar and lexicon. Each word in the domain vocabulary is assigned to one or more of the word classes in which the word participates. Each context becomes a grammatical pattern which is constrained by the word classes that can occur in the pattern; that is, the word classes become selectional restrictions that must be met for a given grammar rule to apply. For example, in SUBJECT-VERB-OBJECT patterns, certain verbs in the sublanguage will take only certain kinds of subjects and objects, and a specialized S-V-O rule can be coded to capture these specialized constraints.

According to Marsh (1983 and 1986), ill-formed input in sublanguages has several features. First, the ill-formed input is made up of relatively few constructions; second, the errors consist of sentence fragments, each of which is part of some rule of proper grammar; third, sentence fragments can be related to full grammatical forms based on the elements which have been deleted. When a fragment is recognized as an instance of a given pattern, and when that pattern has selectional restrictions based on the semantic classes of the sublanguage, then it is possible to determine the semantic class of the deleted items. Hence error recovery, while not recovering a particular lexical item, can at least recover the semantic class of the deleted element.

Our approach to ill-formed input shares some of the benefits of the sublanguage approach. In particular, it is possible to recover the semantic class of a deleted item. This is achieved in two ways. First, the domain knowledge will often supply the necessary information; and second, the matching of a grammar rule in which one of the constituents is missing will enable us to recover the class of the item that has been deleted.

Our approach is perhaps more general than the sublanguage approach. This is because we do not need any special grammar rules to capture the ill-formed constructions of a corpora; because we can handle errors which are *not* simply deletions of a proper construct (for example, 'engine stalls intermittent'); and because our method does not depend on the existence of a sublanguage within the corpora. It may turn out that the sublanguage approach provides more powerful error recovery procedures with a corpora that contains a sublanguage, but that our approach works better when no such sublanguage exists.

Directions of Further Research

ULINK is being expanded in two directions. First, a system that uses syntactic methods to recover from errors is being implemented so that it can be compared to ULINK.

The comparisons will cover both the efficiency of the parse and the percentage of errors covered by the two systems. One interesting possibility is that our approach will recover from some errors better than the syntactic approaches, while syntactic methods will work better for other errors. For example, our approach works well for terse text, in which many words have been left out and in which morphological changes that lengthen words are often ignored. On the other hand, it is not clear how well our approach will handle sentences with extra words - that is, garbage - embedded in them. It may be that syntactic approaches handle this problem better, though the issue is far from settled. The second direction is to test ULINK in another domain, both to see how well the system covers the grammatical errors of that domain, and to see how long it takes to encode the knowledge of the second domain into the representations needed by ULINK. The outcome of these two efforts will help to determine whether a semantics-driven approach to ungrammatical input is more effective than other approaches have been.

References

Grishman, R., Hirschman, L., and Nhan, N. T., (1986) Discovery Procedures for Sublanguage Selectional Patterns: Initial Experiments. In *Computational Linguistics*, Vol 12, No. 3, July-September 1986.

Grishman, R., and Kittredge, R. eds. (1986). *Analyzing Language in Restricted Domains: Sublanguage Description and Processing*. Hillsdale, New Jersey:Lawrence Erlbaum Associates. 1986.

Hirst, G. (1983). Semantic interpretation against ambiguity. PhD Thesis, Brown University, Department of Computer Science, Providence RI, Research Report #CD-83-25.

Jensen, K., Heirdorn, G. E., Miller, L. A. and Ravin, Y., (1983). Parse Fitting and Prose Fitting: Getting a Hold on Ill-Formedness, AJCL Vol 9, Nos 3-4, 1983.

Kittredge, R., and Lehrberger, J. eds. (1982). *Sublanguage: Studies of Language in Restricted Semantic Domains*. Berlin:Walter de Gruyter. 1982.

Lytinen, S. (1990) Robust processing of terse text. In *Proceedings of the 1990 AAAI Symposium on Intelligent Text-based Systems*, Stanford CA, March 1990, pp. 10-14.

Lytinen, S. and Roberts, S. (1989). Lexical acquisition as a by-product of natural language processing. In *Proceedings of the First International Lexical Acquisition Workshop*, IJCAI-89, Detroit, MI, August 1989.

Marsh, E. (1983) Utilizing domain-specific information for processing compact text. In *Proceedings of the Conference on Applied Natural Language Processing*, Santa Monica, CA, pp. 99-103.

Marsh, E. (1986) General Semantic Patterns in Different Sublanguages. In *Analyzing Language in Restricted Domains: Sublanguage Description and Processing* (Grishman and Kittredge, eds.). Hillsdale, New Jersey:Lawrence Erlbaum Associates.

Mellish, C. (1989). "Some chart-based techniques for parsing ill-formed input." In *Proceedings of the 1989 Conference of the Association of Computational Linguistics*. Vancouver, B.C., June 1989.

Shieber, S. (1986). *An Introduction to Unification-based Approaches to Grammar*. Center for the Study of Language and Information, Menlo Park, CA.

Weischedel, R. M. and Sondheimer, N. D. (1983) Metarules as a Basis for Processing Ill-Formed Input. In *American Journal of Computational Linguistics*, Vol 9, Nos 3-4, 1983.

Winograd, T. (1972). *Understanding Natural Language*. New York: Academic Press.

Woods, W. A., Optimal Search Strategies for Speech Understanding Control. In *Artificial Intelligence* 18 (3), 1982.