INTEGRATED PROCESSING PRODUCES ROBUST UNDERSTANDING

Mallory Selfridge

Department of Electrical Engineering and Computer Science
The University of Connecticut
Storrs, Connecticut 06268

Natural language interfaces to computers must deal with wide variation in real-world input. This paper proposes that, in order to handle real-world input robustly, a natural language interface should be constructed in accord with principles of integrated processing: processing syntax and semantics at the same time, processing syntax and semantics using the same mechanisms, and processing language and memory using the same mechanisms. This paper describes an experimental natural language interface constructed according to these principles which displays the desired robustness. The success of this interface suggests that future real-world interfaces could achieve robustness by performing integrated processing.

1 INTRODUCTION

Natural language interfaces to computers must deal with wide variation in real-world input. Since real-world input is often missing words and contains variant syntax, a useful natural language interface must understand such input. Unfortunately, the technology needed to provide this robustness is not fully mature, and there is uncertainty as to the directions in which such maturity lies.

Part of this uncertainty centers around the relationship between syntax, semantics, and world knowledge in natural language processing. One theoretical position, the integrated processing hypothesis (Schank 1981), describes an approach to computer modelling of human language processing, based on the idea that these three sorts of knowledge must be applied together and interactively. Since human language understanding is robust, a computer model of human language processing based on this position should also be robust.

This paper describes a research project designed to explore this conjecture, and describes a robust natural language interface, called MURPHY, which embodies the integrated processing hypothesis. It argues that integrated processing yields robustness and that this hypothesis therefore represents a promising approach to the construction of robust natural language interfaces. MURPHY has been developed within a limited domain, and questions remain about the generality of its techniques. Nonetheless, its performance within this domain suggests that generalization to a richer and more realistic domain is possible.

This paper first defines the term **robust** as it will be used here, and introduces the MURPHY system. Second, it considers previous work on the problems of robustness. Third, it describes the integrated processing hypothesis and the motivation for the research strategy adopted here. Fourth, it describes the MURPHY system and its performance in detail, and then argues that this performance derives from its implementation of the integrated processing hypothesis. Finally, it suggests that the integrated processing hypothesis is indeed a promising approach to the construction of robust natural language interfaces, and that MURPHY represents a successful first step.

2 ROBUST UNDERSTANDING

2.1 WHAT IS ROBUSTNESS?

The broadest possible definition of **robustness** on the part of a natural language interface would involve a language ability equal to or greater than that of a human; clearly this is too ambitious. Instead, the term **robust** will be used here to refer to a particular subset of human language abilities. This subset includes first the ability to understand utterances that are missing various words, and that contain words out of their grammatically preferred order.

0362-613X/86/020089-106$03.00

Second, if an input is initially misunderstood, a robust natural language interface should converge on the correct understanding by continuing to infer the next most likely meaning based on the input and context. Finally, a robust natural language interface must be able to use corrections provided by the user to direct the inference of the next most likely meaning. Since the output of the understanding process in such an interface must be a representation of the meaning of the input, such an interface would thus be limited primarily by its semantic knowledge of the domain of discourse; it could not understand an utterance if the meaning of that utterance was not representable within its knowledge. Otherwise, it would eventually understand. These aspects of robustness are considered in more detail below.

There are a number of situations in which an utterance can be missing words. First, users usually omit words whose meanings can be inferred from the general context by the listener. Second, the user can be deliberately employing ellipsis, intending the listener to complete his understanding using concepts drawn from conversational history. Third, the utterance can contain unknown words, which are "missing" as far as understanding is concerned. Finally, since no one has demonstrated a perfect technique for predicting which words are or are not going to be missing, a robust natural language understander must be prepared to understand utterances with *arbitrary* words omitted, albeit perhaps taking longer to converge in difficult cases.

In addition to missing words, one cannot guarantee that the input will be syntactically well formed. Rather, it will sometimes be ill formed in various unpredictable ways. This paper will be concerned only with the simplest type of variant syntax, consisting of words positioned incorrectly within the utterance. Note that the mispositioning can result in an input that is meaningful, even though this meaning is not what was intended. Further, the words can be correctly positioned with respect to the unintended utterance, even though they are mispositioned with respect to the intended utterance. This issue is considered in greater detail later.

Under some conditions, however, any natural language understander will misunderstand, since alternative interpretations may be equally preferable. A robust interface should address this problem by verifying its understanding with the user, and being prepared to "guess again" if it proved incorrect. Such an interface should be prepared to generate first its most likely interpretation, then its next most likely, and so on, until no further possibilities exist. This would *guarantee* that the interface will eventually understand the utterance, assuming, again, that the meaning of the utterance is within its domain of expertise. Note that the capacity to systematically exhaust the possible meanings of an utterance is beyond the ability of human understanders. A human can produce many interpretations, but cannot be guaranteed to generate all possible interpretations due to factors such as memory limitations. Notwithstanding the current impossibility of equalling or exceeding the overall language ability of humans, a robust understander should, if possible, exceed human performance in this particular regard, if possible.

This ability to infer the next most likely meaning of the input is important but incomplete. If the listener incorrectly infers the meaning of the speaker's utterance, the speaker does not usually say "no" and wait for the listener to infer a second meaning. Rather, the speaker usually supplies a correction. It is thus appropriate that a robust natural language interface not only be able to conjecture the next most likely interpretation but also that it be able to use corrections from the user when they are supplied.

A robust understander should also display a number of other desirable characteristics beyond those discussed above. For example, it must be able to handle words with multiple meanings. Further, it should be able to understand despite false starts, irrelevant interjections, and unknown words. Third, it should provide spelling correction and related support. Fourth, it should have a mixed-initiative conversational ability. Finally, it should be able to learn new word meaning and syntax. Such characteristics are beyond the scope of this paper. However, section 8 argues that MURPHY possesses some of these characteristics as well.

Thus, for the purposes of this paper, a robust understander is one that can be guaranteed to eventually understand input utterances despite arbitrary missing and out-of-order words, both with and without corrections, and including ellipsis, on the basis of semantics and syntax, domain knowledge, and context.

2.2 A ROBUST UNDERSTANDER

In order to address the problem of robust understanding, the MURPHY system was developed. MURPHY operates in conjunction with a robot assembly system (Engelberg 1983; Engelberg, Levas, and Selfridge 1984; Levas 1983; Levas and Selfridge 1984; Selfridge 1983). Both MURPHY and the robot assembly system are written in Franz Lisp on a VAX-11/780. MURPHY allows a user to question and direct the robot assembly system using natural language. Natural language can be used to specify low-level image operations, ask high-level questions about the relationships between objects in the image, describe the appearance of unknown objects for future use by the system, and teach the manipulator how to perform new assembly tasks.

The following example illustrates MURPHY's robustness. It concerns the recognition and assembly of three components of a simple electric switch – the base, the contact, and the post – into the completed switch. MURPHY knows the meaning of each word in this example, and has appropriate domain knowledge. Specifically, it knows that each component is a physical object that is part of the switch. Further, it has geometric models of each object required for planning robot motions. Finally,

it has knowledge about how the switch base and contact appear in the image. In order to demonstrate MURPHY's ability to learn new visual knowledge, however, the knowledge of how a switch post appears in the image has been removed, and the knowledge of how to translate the geometric model of the post into appearance information has also been removed. As will be seen, this is why the system can talk about the switch post without being able to visually recognize it, and allows MURPHY's abilities to be easily demonstrated. The user's inputs are shown in lower case, MURPHY's responses in upper case, and parenthetical notes have been added to describe system operation.

(1) Display the current image workspace.
DO YOU MEAN "Display the current workspace image"?
>Yes.
OK.
(The system displays the image.)

(2) >Threshold the image at fifty, and then apply boundary following for regions larger than thirty-five pixels.
DO YOU MEAN "Threshold the current workspace image at fifty and then apply boundary following for regions larger than thirtyfive pixels in the current workspace image"?
>Yes
OK.
(The system displays the operations occurring.)

(3) >What is in the image?
A SWITCH BASE AND A SWITCH CONTACT.
(The base and contact are outlined and labeled.)

(4) >Where is the switch?
DO YOU MEAN "Where is the switch post"?
>No, contact.
THE SWITCH CONTACT IS NEXT TO THE SWITCH BASE.

(5) >Is there a switch post?
WHAT DOES A SWITCH POST LOOK LIKE?

(6) >A metal cylinder two inches long.
DO YOU MEAN "A switch post is a metal cylinder two inches long"?
>Yes.
THERE IS A SWITCH POST NEXT TO THE SWITCH CONTACT.
(The post is outlined and labeled.)

(7) >Make a switch.
OK
(High-level planning drives the two manipulators to assemble the base, post, and contact into a completed switch.)

In this example, MURPHY enabled a user to interact with the robot assembly system to assemble the electric switch. The user first specified certain low-level image-processing operations, and then asked a high-level question that prompted the system to perform additional bottom-up and top-down recognition processing in order to generate an answer. The user then asked for additional information, which was supplied. Next, the user asked about the existence of an unreported object. The system displayed mixed-initiative capabilities by responding with a query about the appearance of that object. Instead of supplying a description of the object's appearance, the user answered with a description of the object's shape. The system then inferred the appearance of the object, reexamined the low-level image description, found the object, and answered the user's original question. Finally, the user commanded the system to assemble the parts into a complete switch, which it did.

Of specific importance to this paper, however, is the fact that in this interaction MURPHY displayed exactly the kind of robustness described in section 2. In (1) the words *image* and *workspace* were reversed. MURPHY understood correctly despite this variant syntax. In (4), the word *contact* is missing. MURPHY first infers that the missing word was *post,* but is corrected by the user and told that the missing word was *contact.* An additional example of MURPHY understanding despite missing words appears in (2). In (5), the system displays mixed-initiative in response to the user's question "Is there a switch post in the current workspace image?". It answers this question with another question, "What does a switch post look like?". In (6), the user's reply is missing several words, including the primary frame-supplying word. MURPHY infers that the user meant "A switch post is a metal cylinder two inches long". This demonstrates the ability to infer the missing frame-supplying word *is,* and the ability to use conversational history to understand the elliptical reference to "A switch post". In these cases MURPHY meets the criteria established in section 2.1.

Within the robot assembly domain MURPHY currently knows about fifty words and five phrases, and about seventy concepts of fifteen different types. In addition, it has been briefly tested in domains other than robot assembly. During a typical interaction, MURPHY usually responds within five or ten seconds. Occasionally it takes more than a minute on long test sentences. Its response time is acceptable for an initial implementation designed to address theoretical questions, and strongly suggests that with tuning MURPHY could provide responses in real time almost always when in realistic situations.

3 Prior Research on Robustness

Prior research has addressed the problem of robust understanding from a number of different perspectives. Hayes and Mouradian (1981) apply a grammar to utterances flexibly enough to interpret a variety of grammatical deviations. This is done using a bottom-up, pattern matching parser that employs **parse suspension and continuation** to the arcs of an ATN (Woods 1970). Besides optional pattern elements, flexibility is achieved by relaxing consistency constraints and allowing out-of-order matches. Kwasny and Sondheimer (1981) extend Hayes and Mouradian's approach by recording the

nature of the grammatical deviations. Their parser applies grammar relaxation techniques to arcs of an ATN. Whenever an arc of the normative grammar, specifying the structure of a well-formed utterance, cannot be traversed, a **deviance note** is created and the arc is traversed anyhow. This note records how the utterance deviates from the expected grammatical form, and allows parsing to proceed in the presence of variant syntax. Additionally, feature relaxation techniques allow an inappropriate word to stand in place of a correct one.

Both these works suffer from many of the same problems. Although the intent of both is to focus on a specific subset of the overall problem, it is difficult to verify the success of either approach without a semantic component. Second, neither can infer a next interpretation if its initial conjecture was incorrect. Third, neither can handle arbitrary missing words. Finally, the ability of either to handle variant syntax is limited: they can handle variations on only a subset of the total classes of syntax their parser handles.

More recently, Weischedel and Sondheimer (1983) describe work that significantly extends some of the ideas reported by Kwasny and Sondheimer (1981). Their extension involves the use of **meta-rules** to deal with ill-formed input. These meta-rules are intended to recognize an instance of ill-formedness and prescribe actions that may provide understanding. Although the approach of using meta-rules appears to handle well-formedness and appears worthwhile, two characteristics distinguish the meta-rule approach from the present research. First, Weischedel and Sondheimer address themselves only to the problem of processing ill-formed input, and leave for later research the problem of integrating this approach with techniques for handling other aspects of robustness. Second, and more important, their approach to handling ill-formed input differs computationally from the parsing mechanism it overlays, while the research reported here explores the processing of ill-formed input using the same mechanism as that used for well-formed input.

Hayes and Carbonell (1981) report research closer to that described in this paper. Their work combined a number of different approaches within two different experimental parsers. CASPAR combined a search for a semantic case frame with a linear pattern matcher to build a representation of the meaning of the input. DYPAR combined a context-free semantic grammar, a partial pattern matcher, and equivalence transformations for building a representation of the meaning of the utterance. [Note that the DYPAR program described by Hayes and Carbonell (1981) is entirely different from the DYPAR program described by Dyer (1982).] While both appear to incorporate promising techniques, neither CASPAR nor DYPAR displays a high degree of robustness. While CASPAR can handle

- unexpected and unrecognizable interjections in the input,
- missing case markers,
- out-of-order cases, and
- ambiguous cases,

it cannot

- understand if the word whose meaning builds the primary semantic case frame is missing,
- guess again,
- handle ellipsis, or
- understand utterances with arbitrary out-of-order words and missing words.

DYPAR seems similarly limited. Although it is embedded in an interesting database management system, the degree of robustness it displays is not clear.

Carbonell and Hayes (1983) describe research extending that reported by Hayes and Carbonell (1981). They describe a number of "recovery strategies" that can enable understanding to proceed in the presence of what are termed "extragrammaticalities", and which are tested using CASPAR, DYPAR, and a parser called DYPAR-II. Although much of the approach taken is similar to that described here, much of it represents an alternative approach to solving similar problems that focuses on a number of difference processing mechanisms rather than on a single, integrated mechanism as described in the following section. Furthermore, no single program appears to use all the strategies described by the paper, and thus the utility of the strategies taken over-all is difficult to assess. Finally, none of these programs appear capable of continuing to generate alternative interpretations of an input until confirmed by the user; they are not guaranteed to eventually understand the input.

Understanders built within other paradigms have also displayed various degrees of robustness. What might be termed "semantics-oriented" understanders have displayed high performance understanding, producing from an utterance a representation of the meaning of that utterance. For example, ELI and SAM (Riesbeck and Schank 1976, Cullingford 1978), CA (Birnbaum and Selfridge 1981), and ACE (Cullingford, Krueger, Selfridge, and Bienkowski 1981) are similar in spirit to MURPHY, in that each attempts to combine word meanings into a representation of the meaning of the utterance as a whole, and then allow later memory processing access to this understanding. However, these programs can best be thought of as demonstrating the power of memory-based understanding while failing to fully exploit the potential of integrated processing. Each are relatively intolerant of missing words and variant syntax, although each has various abilities in these respects. Other approaches, such as the NOMAD system (Granger 1984), and those reported by Wilks (1976) and Fass and Wilks (1983), are also related to the approach taken in this paper. However, these systems differ significantly in their approach to robustness. For example, while the NOMAD system does present alternative interpretations of an imperfectly understood input, and does employ syntactic knowledge and world knowledge simultaneously during understanding, it is not guaranteed to eventually arrive at the intended meaning of an input (given

NOMAD's domain, this would be difficult in any event because input utterances do not originate with the user) and its language processing and memory processing do not appear to employ the same mechanism. Similarly, systems described by Wilks (1976) and Fass and Wilks (1984), while similar in their use of preferences to MURPHY, are not guaranteed to always eventually understand and employ different mechanisms for language and memory processing.

Finally, it is important to consider high-performance knowledge-based understanding mechanisms, such as described by Dyer (1982) and Lebowitz (1980). These programs demonstrate impressive understanding abilities in the domains of understanding complex stories about interpersonal relationships and news stories about terrorism, respectively. They convincingly demonstrate the power of high-level memory processing in difficult understanding tasks. However, neither has concentrated on the question of robustness as the term is being used in this paper. A final answer to the question of robustness will certainly incorporate such high-performance memory processing.

Each of the systems described in this section has certain robust aspects, but each leaves something to be desired. While it is possible to imagine extending some of these systems to remove various limitations, it is impossible to judge the success of such extensions in the absence of actual implementations; no evaluation can be made on the basis of such hypothetical extensions. Thus, no previous research has developed a natural language understander that is robust in all the ways being addressed here.

4 The Integrated Processing Hypothesis

In order to build a robust natural language interface, one must specify the relationships between syntax and semantics, and between language understanding and memory processing, because actual construction of an interface requires a commitment to specific relationships.

Schank and Birnbaum (1981) address these issues in proposing the integrated processing hypothesis. Generalizing from their discussion, these issues can be summarized by the following three questions:

Is syntax processed prior to semantics, or
are syntax and semantics processed at the same time?

Is syntax processed separately from semantics, or
are they processed together by the same process?

Are language processing and memory processing
different processes, or
are they fundamentally the same process?

As discussed by Schank and Birnbaum (1981), there are roughly two polar positions on these issues. One position might be called the "separatist" position, while the other can be termed the "integrated" position. Each position can be characterized by its answer to these three questions. The first question concerns the temporal relationship between semantic and syntactic processing during understanding. The separatist position suggests that a syntactic analysis of an utterance is performed prior to any semantic analysis, and that its output is a syntactic description of the utterance. This output is then passed to the semantic analysis process. In opposition to this view is the integrated perspective. This proposes that syntactic analysis is carried out at the same time as semantic analysis. Thus the temporal order between syntactic analysis and semantic analysis in language processing is a matter of disagreement, and must be addressed when constructing a robust natural language interface.

The second question concerns the nature of the mechanisms that process syntax and semantics. The separatist view suggests that the mechanism that constructs a syntactic description of an utterance is a different mechanism from that which builds a representation for the meaning of the utterance. That is, this view suggests that syntactic analysis operates according to a different algorithm than semantic analysis. The integrated view, on the other hand, proposes that syntax and semantics are processed by the same mechanism. This mechanism operates equally well both on syntactic information and semantic information. These two positions are thus quite different, and constructing a robust natural language interface requires a choice.

The third question concerns the relationship between language processing and memory processing. The separatist position is that language processing is a special, specific function, largely unconnected from memory processes. In this view, memory is thought to be a relatively passive entity, with little active processing. The integrated position, however, holds a different view of the role of memory in language processing. It suggests that language processing is primarily a memory-based process, and further takes the position that language processing and memory processing are the same process. This question is of particular importance because a robust interface will presumably have to employ memory processing of some sort.

Note that an intermediate position between the integrated and separatist positions is possible. One can hold the integrated position with respect to one or two questions and the separatist position with respect to the rest. For example, Bobrow and Webber (1980) describe a natural language interface in which syntax and semantics are processed in a logically simultaneous, intermingled fashion, yet in which syntax and semantics are processed by different mechanisms and in which language and memory processing is performed by different mechanisms. Nonetheless, the distinction represented by the two positions is useful.

Schank and Birnbaum's integrated processing hypothesis is basically the hypothesis that the integrated position correctly characterizes human processing, and can be summarized by the following:

Syntax and semantics are processed at the same time.

Syntax and semantics are processed by the same process.

Language processing is fundamentally the same as memory processing.

Schank and Birnbaum present a detailed argument to justify the integrated processing hypothesis, but its primary impact here is its consequences as a model of human understanding. That is, if the integrated processing hypothesis in fact describes human processing, and since humans are robust language processors, then one way to build a robust natural language interface is to incorporate the integrated processing hypothesis into a natural language interface. This conjecture might be termed the "integrated processing produces robust understanding conjecture", or the IPPRU conjecture.

It is important to understand what this conjecture does *not* say. The IPPRU conjecture does not claim that embodying the integrated processing hypothesis is *necessary* to produce robust understanding, only that it is *one* approach that does work. Although establishing the necessity of the integrated processing hypothesis to robust understanding would be desirable, this is not within the scope of this paper. Rather, the research reported here concerns a first step to the later establishment of necessity. Neither the Integrated Processing Hypothesis nor the IPPRU conjecture claim that syntactic knowledge is the same as semantic knowledge. While syntax is processed at the same time as semantics, and by the same mechanism, this paper proposes a different breakdown between syntax and semantics. This breakdown is *knowledge-based* instead of *processing-based.* That is, the difference between syntax and semantics in this view lies in the specific knowledge each represents rather than the order or processing mechanisms of each.

Evaluating the IPPRU conjecture involves certain questions. How best can the integrated processing hypothesis be embodied in a program? What should the domain of that program be? How can its performance be evaluated? In order to address the question of embodying the integrated processing hypothesis within a program, an important distinction must be made between

- a program that may have several modules but which *embodies the integrated processing hypothesis* by virtue of the algorithms it employs and the manner in which it manipulates its data, and
- a program that not only embodies the integrated processing hypothesis but which also is *itself integrated,* in the sense of being non-modular.

Ideally, the integrated processing hypothesis should be embodied in a program that is actually integrated as well. However, the construction of such a fully integrated program is a lengthy process and requires a number of difficult design decisions. In order to gain information on which these design decisions can be made, the MURPHY system was developed as a *rapid prototype,* which, although not fully integrated, does embody the integrated processing hypothesis. Thus, MURPHY's performance does bear directly on the IPPRU conjecture, even though MURPHY itself is not fully integrated.

The second question concerns the domain within which a natural language program operates. Ideally, the domain will be both large and realistic. However, since effort on a large and realistic domain must be justified by high performance within a limited domain, it is appropriate to experiment with such a limited domain as a necessary first step to a large and realistic domain. This is the approach taken by others within this area of research (e.g. Hayes and Mouradian 1981. Kwasny and Sondheimer 1981, Hayes and Carbonell 1981, Dyer 1982, Lebowitz 1980). The limited semantic domain chosen for this research is that of small-scale robotic assembly in a laboratory context. This domain is appropriate because it is a subset of a potentially useful real-world domain and because it provides a measure of understanding – the degree to which the system successfully carries out commands, answers questions, and remembers and uses declaratives.

The third question concerns the criteria for evaluating the research. Under what conditions will it be considered a success? There appear to be basically two: First, does it in fact perform robustly within its domain? That is, does it fulfill the requirements described in section 2? Second, does it provide insight as to how its techniques might be applicable both to an expansion of the existing domain and to other domains? That is, are its limitations clear, are the areas in which research is needed apparent, and is there suggestive evidence that such additional research would be successful? Positive answers to both these questions would suggest that this research should be considered successful.

5 MURPHY's Architecture

This section describes the MURPHY system in detail. MURPHY is composed of four major component programs:

- A natural language analyzer (NLA), which accesses a dictionary of words and phrases to perform low-level understanding of the words in the utterance;
- An inferencer (Robust Back End, or RBE), which completes understanding using conversational history, context, and a body of domain knowledge;
- A conversational control program (CCON), which performs inference on the input meanings of the user's utterances and provides a mixed-initiative conversational ability, and which allows MURPHY to interact with the robot assembly system;
- A natural language generator (Conceptual Generator, or CGEN), which accepts concepts and expresses them in English.

A user's utterance to MURPHY is analyzed by NLA as far as possible. NLA then passes its understanding of the utterance to RBE, which completes the understanding process using domain knowledge and the conversational history. RBE then verifies its understanding with the user,

and if incorrect it infers the next most likely meaning, and so on until it either infers the intended meaning or exhausts the possibilities. If the latter, control is returned to NLA to produce its next most likely understanding of the utterance, which again is passed to RBE for inference. When the intended meaning is confirmed by the user, it is passed to CCON, which uses test-action rules to infer a response to the utterance. Some responses involve queries or answers directed to the user, some involve internal inferences, and some direct calls to the robot assembly system. Throughout, CGEN is used when needed to generate natural language responses.

Almost every component of MURPHY is relevant to the question of robust understanding; NLA, RBE, CCON, the dictionary, context, and domain knowledge all have important roles. This section first describes each in turn, and then describes how these components embody the integrated processing hypothesis.

5.1 REPRESENTING DOMAIN KNOWLEDGE AND CONTEXT

MURPHY's domain knowledge consists of a set of semantic primitives appropriate to the domain, represented in Conceptual Dependency format (Schank 1975, Schank and Abelson 1977; although MURPHY could be implemented with any of a wide variety of knowledge representation formalisms generally similar to Conceptual Dependency). Each semantic primitive, henceforth called a CD, consists of a **header** followed by a set of labelled **slots.** Together, the header and labelled slots comprise a CD **frame.** CDs can be combined with one another by placing one CD into a slot of another, according to certain restrictions: each CD has certain properties, and each slot in a CD can only accept other CDs with certain properties. For example, ⟨requires human⟩ specifies that the slot filler is required to have the property ⟨human⟩. In addition, certain properties are preferable, but not essential. For example, ⟨prefers small⟩ specifies that a filler that has the property ⟨small⟩ is preferable to one which does not; however, slot filling can still proceed even if the preferred property is absent. Note that this notation for restrictions on what CD can fill a slot in what other CD is not intended to be fully adequate but only to satisfy current needs. In a real-world system the use of simple concept attributes would prove insufficient. A real world domain requires complex reasoning to determine if a concept should be combined with another concept. However, it seems reasonable to conjecture that a rich system can be built around the idea that when such reasoning is completed its result will be an assertion similar to the current attributes. Thus, the current approach does not rule out the use of complex reasoning, and is upwardly compatible with such reasoning. However, the current limited domain requires only the existing simple predicates. Thus, the primary definition of a CD consists of the frame definition, the properties of the CD, and the restrictions that specify which kinds of CDs can fill each slot. For example, the following shows a definition for the CD that refers to the switch contact, object1:

define-concept

frame-header:	object1	
isa:	physicalobject	
frame:	(object1 partof (nil) ref (nil))	
slot-restrictions:	partof	requires physicalobject
	ref	requires determiner

In addition to the information captured by definitions of this type, MURPHY's knowledge of the various CD predicates is also represented in a more distributed fashion throughout the system. For example, MURPHY also knows object1 as a three-dimensional object in space, represented geometrically as a number of points defining the vertices of a planar solid. This representation is used by robot path planning and collision avoidance software, and is an essential part of the meaning of object1. The other kind of additional knowledge MURPHY possesses about its meaning representation is possible inferences within the conversational control. Each conversational control rule matches some configuration of CDs, in order to implement inferences which may be drawn from the presence of a certain CD. Each rule thus encodes additional knowledge of a CD.

5.2 REPRESENTING LANGUAGE KNOWLEDGE

MURPHY's knowledge of words is contained in a dictionary. A word definition consists of the word's meaning and its syntax. The meaning is a single CD or a complex of nested CDs from domain knowledge. That is, MURPHY's word meanings are pointers into its knowledge of the world. Each word meaning may have several empty slots. For each empty slot, the word definition includes syntactic knowledge about where in the utterance a slot filler is expected to be. This syntactic knowledge is expressed as a set of independent **syntactic features.** These features are formed from the positional predicates PRECEDES and FOLLOWS, which are applied to the short term memory around which NLA's processing focuses. This short term memory contains, in order, the input words, their meanings, and the slots these meanings fill in other meanings. PRECEDES and FOLLOWS relate the position in the input of a potential slot filler to either the meaning containing the slot, a filler of another slot in that word's meaning, or a lexical function word. To represent the knowledge that the filler is found following the meaning containing the slot, the word definition includes the predicate "follows parent" indexed under that slot. Similarly, the predicate "precedes (slot object)" represents the knowledge that the filler is found preceding the filler of the ⟨slot name⟩ slot, and "follows (fw ⟨function word⟩)" represents the knowledge that the filler is found following the function word ⟨function word⟩. Several predicates are used to completely describe the position of a filler in an utterance. Thus, each slot in a word meaning has associated with it a collection of features describing where in the

utterance a filler is expected to be. For example, in the definition of the word *contact* (as in *the switch contact*) there is a CD that represents its meaning, and a collection of syntactic features specifying where the fillers of the empty slots are expected to be.

define-word
word: contact
meaning: (object1 partof (nil) ref (nil))
syntax: partof-filler precedes parent
follows ref-filler
ref-filler precedes parent
precedes partof-filler

Most likely, this representation of syntax cannot hope to encompass an entire natural language. It is used here because it is powerful enough to describe the syntax of the natural language capabilities of interest. However, it has demonstrated reasonable expressiveness in a number of different applications (Birnbaum and Selfridge 1981, Cullingford, Krueger, Selfridge, and Bienkowski 1981; Selfridge 1980, Selfridge 1981a; Selfridge 1981b; Selfridge 1982) and thus its use here is not entirely ad hoc.

5.3 THE NLA PROGRAM

When the user types an utterance to MURPHY, the utterance is first processed by the NLA program. NLA is a descendant of the CA program (Birnbaum and Selfridge 1981), and also uses concepts derived from Wilks (1976). Its role in the understanding process is to create as complete a CD representation of an utterance's meaning as possible using only the meanings of the words in the utterance. NLA's processing centers around a short-term memory called the C-LIST. During analysis, the meaning of each input word is placed on the C-LIST (currently, NLA is limited to words with only a single meaning; it cannot disambiguate among multiple words senses). The syntactic and semantic features associated with slots in the meanings of each word on the C-LIST are then checked to see if the meaning of any other words on the C-LIST can fill any of them. If so, the CD that most satisfies the syntactic and semantic features associated with a particular slot is placed in the slot. This process is repeated for each CD on the C-LIST. When completed, what remains on the C-LIST is one or more CDs constructed by combining the meanings of the words in the utterance. The CD or CDs represent as much understanding of the utterance as could be achieved by examining only the meanings of the utterance words. More formally, NLA's basic algorithm is as follows:

(1) Place the CD meaning of each utterance word or phrase on the C-LIST.
(2) For each empty slot in each CD on the C-LIST, collect the syntactic and semantic features associated with that slot.
(3) Search the C-LIST, and retrieve all the CDs that satisfy that slot's semantic requirements. These CDs are candidate slot fillers.
(4) Order the candidates by preference value (the number of semantic preferences and syntactic features a candidate satisfies).
(5) Examine the candidate with the highest preference value. If the CD is not marked "used", then fill the current slot with it and mark it "used".
(6) If the candidate is marked "used" and its preference value for the slot it already fills is higher than the current preference value, then reject it and examine the candidate with the next highest preference value.
(7) If the candidate is marked "used" and its preference value for the slot it already fills is lower than the preference value associated with that other slot, then remove it from the other slot and fill the current slot with it, and recursively call NLA to refill the other slot.

In addition to the above algorithm, NLA performs additional work. This additional work is needed to allow NLA to produce a next most likely interpretation of the input. It does this by keeping track of all the candidates for each slot and their preference values, and by maintaining a list of rejected interpretations. When called upon to generate the next most likely interpretation, it does so by finding the most preferred interpretation that does not appear on the list of rejected interpretations. In this way, NLA can provide all possible interpretations of the input (in conjunction with the RBE program, described in the following subsection), ranked according to likelihood. It should be noted that the technique of keeping track of rejected interpretations is crude yet functional; future work will address the question of improving this implementation.

Since section 5.6 argues that both NLA and RBE use essentially the same mechanism, an abstract description of each is important. A number of alternative descriptions exist; viewing NLA's understanding process as tree search is best for the current purposes. The interior nodes of this tree represent partial understandings of the input, while leaves represent complete understandings. That is, interior nodes are CDs that contain empty slots, while leaves are CDs that do incorporate every word meaning. The root of this tree is a **start node,** whose descendants are the CDs from the utterance that have empty slots. Given that the search process has a current node, generating the descendants of that node involves choosing an empty slot, retrieving its tests, using those tests to retrieve from the C-LIST all possible candidate fillers, creating a copy of the CD at the current node for each candidate and filling the slot with a candidate filler, and finally building a new node for each such CD. The CD at each new node will thus have one less empty slot. The new current node is then chosen to be the unvisited node whose filler had the highest preference value for that slot. Search proceeds until a leaf is reached in which all possible slots have been filled from CDs on the C-LIST. Since NLA first chooses slot fillers that best

satisfy the syntax and semantics associated with a slot, its processing strategy implements a kind of local best-first search (Nilsson 1971).

The following example describes part of NLA's processing of the sentence *put the post on the base.* In order to illustrate preference, this example focuses on filling the VAL slot in the meaning of *on,* in the middle of processing the input. Of the two candidate fillers available at this point for the VAL slot (the meanings of *post* and *base*) the meaning of *base* is currently being used as the filler of the OBJECT slot. However, preference overrides this prior slot filling, moves the meaning of *base* to the VAL slot, and finds the next best filler for the object slot. The example begins with the state of the C-LIST at that point, then shows the selection of the filler of the VAL slot, and finally shows the selection of the next-best filler for the object slot.

```
C-LIST:    (PTRANS ACTOR (NIL)
                   OBJECT (PHYS-OBJ TYPE (BASE)
                                    PART-OF (NIL)
                                    REF (NIL))
                   TO (TOP VAL (NIL)))
           (REF) -- used
           (PHYS-OBJ TYPE (POST) PART-OF (NIL) REF (DEF))
           (TOP VAL (NIL)) -- used
           (REF)
           (PHYS-OBJ TYPE (BASE) PART-OF (NIL) REF (NIL)) -- used

EXAMINING: (TOP VAL (NIL))

CHECKING VAL SLOT: requires phys-obj
                    follows "on", "put", object-filler
VAL CANDIDATES: (PHYS-OBJ TYPE (POST) PART-OF (NIL) REF (NIL))
                 preference value 1: follow "put",
                (PHYS-OBJ TYPE (BASE) PART-OF (NIL) REF (NIL))
                 preference value 2: follows "on", "put"
PREFERRED FILLER: (PHYS-OBJ TYPE (BASE) PART-OF (NIL) REF (NIL))

REMOVING FILLER OF OBJECT SLOT

RE-CHECKING OBJECT SLOT: requires phys-obj
                          follows "put", precedes to-filler
OBJECT CANDIDATES: (PHYS-OBJ TYPE (POST) PART-OF (NIL) REF (NIL))
                    preference value 2: follows "put"
                                        precedes to-filler
                   (PHYS-OBJ TYPE (BASE) PART-OF (NIL) REF (NIL))
                    preference value 1: follows "put"
PREFERRED FILLER: (PHYS-OBJ TYPE (POST) PART-OF (NIL) REF (NIL))

C-LIST:    (PTRANS ACTOR (NIL)
                   OBJECT (PHYS-OBJ TYPE (POST)
                                    PART-OF (NIL)
                                    REF (NIL))
                   TO (TOP VAL (PHYS-OBJ TYPE (BASE)
                                         PART-OF (NIL)
                                         REF (NIL))))
           (REF) -- used
           (PHYS-OBJ TYPE (POST) PART-OF (NIL) REF (NIL)) -- used
           (TOP VAL (NIL)) -- used
           (REF)
           (PHYS-OBJ TYPE (BASE) PART-OF (NIL) REF (NIL)) -- used
```

The understanding process is complete when the remaining slots in the meaning of *base* have been examined and the REF slot filled with the meaning of the second *the.* At this point, all the "used" concepts are removed from the C-LIST; those concepts remaining constitute NLA's best understanding of the input:

```
C-LIST:    (PTRANS ACTOR (NIL)
                   OBJECT (PHYS-OBJ TYPE (POST)
                                    PART-OF (NIL)
                                    REF (DEF))
                   TO (TOP VAL (PHYS-OBJ TYPE (BASE)
                                         PART-OF (NIL)
                                         REF (DEF))))
```

At this point, NLA has produced its best understanding of the input. Since this understanding contains empty slots, it would require additional processing by RBE. If NLA were asked to produce its next best interpretations, it would reprocess the input for an alternative meaning, and fill the OBJECT slot with the meaning of *base* and the VAL slot with the meaning of *post*.

5.4 THE RBE PROGRAM

When NLA has concluded processing an input, it may not have been fully understood. If the input was missing words, the C-LIST can contain CDs with unfilled slots and several CDs that have not been combined into a single CD. In these cases, RBE is called to complete understanding. When RBE must fill empty slots in a CD, it infers fillers from domain knowledge. When RBE must combine several CDs into a single CD, it searches domain knowledge for a CD whose empty slots could be filled by them. Once it has filled all the empty slots with potentially correct fillers, and has combined all the uncombined CDs into one, the result is a complete understanding of the meaning of the utterance. Before passing this meaning to the conversational control, RBE verifies its understanding is the one intended by the user by generating it in English. If the user disagrees with the interpretation, RBE searches domain knowledge further, produces the next most likely interpretation, and so on until it either infers the intended meaning or has exhausted all possibilities. In general, RBE prefers to fill a slot with a C-LIST element whenever possible. This is because the user presumably included a word in an utterance because its meaning was intended to contribute to the meaning of that utterance. If no C-LIST elements are appropriate slot fillers, RBE searches corrections, conversational history, and domain knowledge. Thus RBE will infer concepts in the order of reasonable likelihood.

RBE's search process could not be guaranteed to terminate if RBE were as just described. Under some conditions, it searches context and domain knowledge for a CD with empty slots, intending to fill them with the results of partial understanding. That is, RBE must infer a CD with a slot that can be filled by a CD currently on the C-LIST. Unfortunately, this process can proceed indefinitely: how does RBE know when to stop inferring increasingly inclusive CDs? The function of RBE's inference provides an answer: RBE stops when it has inferred a CD to which CCON can respond. Since RBE continues the inference process until it fills all the slots in a CD to which CCON can respond, these CDs serve as goals to RBE. Consequently, they are termed **goal concepts.** Now, there are several possible implementations that allow CCON to communicate goal-concepts to RBE. The current implementation is the simplest: annotate goal concepts as such, and have RBE's search be top-down beginning with a goal concept. If the first goal concept chosen proves incorrect, RBE uses the next one, and so on, until it finds the correct one. Future work will explore more sophisticated search strategies.

RBE's algorithm focuses around a data structure called the NODE-list, initially empty, and is given below:

(1) If there is no goal concept on the C-LIST, collect from conversational history and domain knowledge all goal concepts and place them on NODE-list. Otherwise, place all goal concepts from the C-LIST on NODE-list.

(2) For each empty slot within the first NODE-list element, retrieve from context and domain knowledge all the candidate fillers that satisfy that slot's semantic requirements. Order all candidates from user's corrections before those from conversational history, and order all those from conversational history before those obtained from domain knowledge. Within each group, order candidates according to the number of semantic preferences they satisfy.

(3) For each candidate, create a copy of the top of NODE-list and place that candidate into the slot of the copy, and place the resulting CD back onto NODE-list.

(4) If the CD at the beginning of NODE-list has empty slots, go to (2). If the CD has no empty slots, send it to CGEN to query the user whether it is correct or not.

(5) If the user confirms the CD, send it to CCON for response. If not, remove the CD from NODE-list and go to (2).

Just as NLA's processing can be characterized as tree search, RBE's operation can be described the same way. That is, RBE takes the best understanding provided by NLA as the root node, and descendants are nodes at which CDs have additional slots filled. Leaf nodes are those that have no empty slots at all. Generating the descendants of a node involves

- choosing an empty slot;
- retrieving its semantic requirements and preferences;
- retrieving all possible candidates from domain knowledge, conversational history, and user corrections that satisfy the requirements;
- creating a copy of the CD at the current node for each candidate and filling the slot with a candidate filler; and, finally,
- building a new node for each such CD.

The CD at each new node will thus have a formerly empty slot filled. The filler will often have empty slots of its own, and these will be filled in their turn. Infinite regress is stopped at an arbitrary fixed level to assure termination. (An alternate approach would be breadth-first, and would require no such arbitrary limitation.) The new current node is then chosen to be the unvisited node whose filler had the highest preference value for that slot. Search proceeds until a leaf is reached at which there are no empty slots at all. At this point, the understanding represented by the CD at that leaf is generated in natural language, and the user is asked to verify the understanding. If it was correct, then the search is over. If not, then

RBE backs up and continues the search at the point at which it was halted. This process continues until RBE has searched the entire tree of possibilities.

If the intended meaning has still not been inferred, then NLA is called to generate its next most likely interpretation of the words in the input, and RBE begins again to infer fillers for empty slots in this CD. Thus, RBE continues the search for a complete understanding begun by NLA, using essentially the same local best-first searching process to do so; if its search fails, it returns control to NLA to generate its next most plausible interpretation of the input, and RBE again continues the search. Thus, the best-first search processes of NLA and RBE together perform a unified best-first search. This unified search process exhaustively searches the space of possible meanings of the input utterance, ordered by likelihood as described, and will eventually find any finite CD (Nilsson 1971). Since any intended meaning is assumed to be represented by a finite CD, the search performed by NLA and RBE is guaranteed to eventually infer the intended meaning.

The following example illustrates RBE's processing on the understood meaning of *put the post on the base.* RBE begins when it receives NLA's best understanding of this sentence and places it on NODE-list, as shown below.

```
NODE-list: (PTRANS ACTOR (NIL)
                   OBJECT (PHYS-OBJ TYPE (POST)
                                    PART-OF (NIL)
                                    REF (DEF))
                   TO (TOP VAL (PHYS-OBJ TYPE (BASE)
                                         PART-OF (NIL)
                                         REF (DEF))))
```

RBE removes the first CD in NODE-list – the one shown above – and notes that it has an empty ACTOR slot. It retrieves the semantic requirements and preferences for the ACTOR slot, which specify that the filler is required to be an animate being. Since, in this example, there are assumed to be no corrections or conversational history, RBE searches only domain knowledge for such a filler. It retrieves the concepts for itself and that of the user, creates copies of the CD with these as ACTOR fillers, and pushes them onto NODE-list. The copy containing the concept of MURPHY itself is on the front of NODE-list because it was found first in domain knowledge; this order reflects the knowledge that MURPHY is more likely to be the intended actor. The new NODE-list is shown below:

```
NODE-list: (PTRANS ACTOR (MURPHY)
                   OBJECT (PHYS-OBJ TYPE (POST)
                                    PART-OF (NIL)
                                    REF (DEF))
               TO (TOP VAL (PHYS-OBJ TYPE (BASE)
                                     PART-OF (NIL)
                                     REF (DEF))))

           (PTRANS ACTOR (USER)
                   OBJECT (PHYS-OBJ TYPE (POST)
                                    PART-OF (NIL)
                                    REF (DEF))
                   TO (TOP VAL (PHYS-OBJ TYPE (BASE)
                                         PART-OF (NIL)
                                         REF (DEF))))
```

The cycle continues when RBE again removes the first CD and again examines it for empty slots. The next empty slot it finds is the first PART-OF slot. The only possible filler from domain knowledge is the CD (COMPOUND-OBJ TYPE (SWITCH)). This is used to fill the first PART-OF slot, and the resulting CD is pushed back on NODE-list. During the next cycle, the just-modified CD is removed from the top of NODE-list and the second PART-OF slot is found to be empty, and is likewise filled with (COMPOUND-OBJ TYPE (SWITCH)). The resulting NODE-list is shown below:

```
NODE-list: (PTRANS ACTOR (MURPHY)
                   OBJECT (PHYS-OBJ TYPE (POST)
                                    PART-OF (COMPOUND-OBJ
                                                TYPE (SWITCH))
                                    REF (DEF))
                   TO (TOP VAL (PHYS-OBJ TYPE (BASE)
                                         PART-OF (COMPOUND-OBJ
                                                     TYPE (SWITCH))
                                         REF (DEF))))
```

```
(PTRANS ACTOR (USER)
        OBJECT (PHYS-OBJ TYPE (POST)
                         PART-OF (NIL)
                         REF (DEF))
        TO (TOP VAL (PHYS-OBJ TYPE (BASE)
                              PART-OF (NIL)
                              REF (DEF))))
```

At this point the first CD on NODE-list has no empty slots, and furthermore is a known goal concept. RBE calls the generator to ask the user if this was his intended meaning. If the user verifies this as his intended meaning, RBE has successfully completed the understanding process. If not, then RBE removes this CD from NODE-list and proceeds to cycle further. In the case described above, this would amount to RBE inferring that possibly the intended filler of the ACTOR slot was the user rather than MURPHY itself.

When RBE has successfully verified its understanding with the user, it passes this completed understanding to the CCON program, which will, in turn, infer a response to the input. In addition, RBE also updates the conversational history with the concepts that comprise the completed understanding. Since RBE uses candidates from conversational history before those from domain knowledge, it uses potentially more relevant concepts before less relevant ones. In particular, RBE can infer goal concepts as a function of context. Given an identical input utterance, MURPHY will understand it one way in one conversational context, and another way in another context, since it can obtain a goal concept from conversational history.

5.5 THE CCON PROGRAM

The CCON program is invoked when NLA and RBE have understood the input to the user's satisfaction. It uses a set of "if-then" conversational rules to respond to that understanding. The complete CD representing the meaning of the input is placed on a stack, and the rules are checked to see if the test of any matches the CD on top of the stack. If so, the action of that rule is executed. The action can send commands to the robot system, add CD inferences to the stack, query MURPHY's knowledge base, and ask the user questions by sending concepts to CGEN to be generated. Thus, CCON is partly a rule-based system, partly a problem-reduction problem solver, and partly a knowledge-based inference engine. The following algorithm describes its operation:

(1) Begin when a goal concept is placed on the stack.
(2) Find the first rule whose test matches the concept on the top of the stack; if the stack is empty, return control to NLA to seek another user input.
(3) Pop the stack and execute the action of that rule.
(4) Go to 2.

Although CCON does no significant searching as far as the understanding process is concerned, it is consistent to note that it can as well be seen as performing a search process. Since the action of a rule can place concepts on the stack, CCON can in fact traverse a tree of concepts, and thus perform essentially the same process as NLA and RBE.

5.6 HOW MURPHY EMBODIES THE INTEGRATED PROCESSING HYPOTHESIS

It is important now to consider the way in which MURPHY embodies the integrated processing hypothesis. Three questions must therefore be addressed. First, how does MURPHY process syntax and semantics simultaneously? Second, how are syntax and semantics processed by the same mechanism? Third, how is language processing fundamentally the same as memory processing? These questions are addressed in turn.

To understand how MURPHY processes syntax and semantics simultaneously, consider how NLA processes input. When NLA is understanding the input as well as possible, it is filling slots in one word meaning with another word meaning. To determine the degree to which a filler is appropriate for a slot, NLA performs a set of tests on the potential filler. These tests derive from the slot, and include semantic requirements, semantic preferences, and syntactic features. As described earlier, NLA first collects all the tests, and then evaluates them together. No distinction is made between semantic and syntactic tests during the evaluation process, and no test's output depends on the result of any other test. Since they are independent, the tests are logically simultaneous, and thus NLA processes syntax and semantics simultaneously.

It could be argued, however, that since RBE uses semantic information too, and does so after syntax has been used by NLA, this later use of semantics constitutes a departure from the simultaneity of syntactic and semantic processing. However, at this point language knowledge is not being used, since fillers cannot be found from the C-LIST. Thus it is not appropriate to consider syntactic knowledge, since nothing but input could usefully be processed syntactically. Furthermore, since RBE sometimes returns control to NLA, syntactic processing can be seen as interspersed with semantic and memory processing, and thus embodying a form of simultaneity of syntactic processing and semantic processing. Nonetheless, unifying NLA and RBE to provide for complete simultaneity would be desirable, and remains for future research.

To understand how syntax and semantics are processed by the same mechanism, consider again NLA's processing of the input. When syntax and semantics are being processed by NLA, not only are they being processed simultaneously but the same mechanism within NLA is also performing the processing. This mechanism is the one that takes each feature test in turn, regardless of

whether it is a semantic requirement, semantic preference, or syntactic feature, and evaluates it with respect to a candidate filler to determinate whether to fill the slot with that filler or not. Thus NLA applies the same mechanism to evaluate syntax as semantics. Furthermore, RBE uses this same mechanism when inferring CDs from context and domain knowledge. It retrieves candidate slot fillers based on the degree to which they satisfy semantic requirements and preferences, using exactly the same mechanism as NLA uses to evaluate candidate slot fillers from the C-LIST. It is this mechanism, used by both NLA and RBE, that processes both syntax and semantics in the same manner.

To understand how language processing is fundamentally the same as memory processing within MURPHY, compare the algorithms being executed by NLA and RBE. Within MURPHY, NLA performs "language processing" while RBE performs "memory processing". As described above, each uses essentially the same form of best-first search, and together they combine to form a single unified best-first search. Further, each are manipulating the same CDs in the same way, and control passes back and forth between them. NLA chooses to fill a slot in a CD with a candidate filler CD if that candidate filler satisfies the most syntactic features and semantic requirements and preferences of the CDs in the C-LIST. RBE chooses to fill a slot in a CD from candidate fillers drawn from conversational history and domain knowledge if that candidate filler satisfies the most semantic requirements and preferences of the CDs retrieved. Both NLA and RBE determine the most likely filler for a slot, and hence ultimately the most likely interpretation of the input, by choosing the filler that satisfies the greatest number of syntactic and semantic features and preferences. Thus, both language and memory processing are carried out by the same mechanism of search, evaluation of features, requirements and preferences, and choice of the CD satisfying the greatest number.

6 Robust Understanding

It is important to analyze the relationship between MURPHY's performance and its implementation of the integrated processing hypothesis, and to assess that performance overall. This section will consider the relationship between each aspect of robustness and each component of the integrated processing hypothesis, and will also discuss two general measures of performance.

6.1 Understanding Input with Variant Syntax

To describe how MURPHY understands input with variant syntax, consider again interaction (1) from the example of section 2. In this interaction, the user typed *Display the current image workspace,* in which *image* and *workspace* are reversed. To understand this, NLA combines the meanings of the words in the utterance together, as well as possible, by evaluating the semantic and syntactic features of each slot in a word's meaning with respect to the remaining meanings, which are candidate slot fillers, and choosing as a filler the meaning that satisfies the most features. Even though in the user's utterance the word *workspace* was out of position, and hence not all features were true, its meaning still satisfied the most features, and was hence chosen as the filler of the SOURCE slot. At this point, NLA has produced a single CD containing no empty slots. This CD is verified to be a goal concept, and understanding is complete.

Now, in interaction (1) a filler was out of position with respect to the meaning that contained the slot. Consider the following more complex example not contained in the example on page 91.

> The display current workspace image.
DO YOU MEAN "Display the current workspace image"?
> Yes.
OK (The system displays the image.)

In the user's utterance the word *the* is out of position. However, the system correctly understands that the meaning of *the* is intended to fill the REF slot in the meaning of *image,* and demonstrates its knowledge of the correct position in its query to the user. Correct understanding occurs because although not all of the syntactic and semantic features associated with the REF slot are true, the meaning of *the* is nonetheless the best filler available. Syntactic features are retrieved from the definition of *image,* and from the definition of *display* as well, since the meaning of *image* is understood to fill the OBJECT slot of the meaning of *display* and MURPHY propagates syntax downward from parent to filler. These features are then evaluated with respect to the meaning of *the.* Although not all syntactic features are true, MURPHY fills the REF slot with the meaning of *the* as the best available. These features and their evaluations are shown here:

origin of feature	feature	value
meaning of *image*	ref requires ref-spec	T
definition of *image*	ref-filler precedes parent	T
	ref-filler precedes subject-filler	T
	ref-filler precedes time-filler	T
definition of *display*	ref-filler follows meaning of *display*	F

Each of the examples considered above assumes that the utterance containing the out-of-order words does not have an alternate interpretation for which the words are not out of order. For example, in *Display the current image workspace,* it was assumed that *workspace* was *not* something that could be displayed. If it was, then the utterance makes sense as it is, and MURPHY would have to decide which interpretation was correct, the one in which it was the workspace being displayed or the one in which the image is to be displayed. As far as MURPHY is concerned, there are two distinct situations in which an utterance with variant syntax can have two different meanings. MURPHY can handle one, while the other is beyond its current capabilities. The first is one in which there are two empty slots, each of which can accept the

other's filler. In this situation, MURPHY will assume that the most preferred interpretation is correct, even if the user mispositioned words and actually intended the other. If the user objects, however, MURPHY will infer the alternate interpretation as the next most preferred meaning. The second situation arises when the word that is out of position has multiple meanings, and while it is out of position with respect to its intended meaning, it is in a correct position with respect to an alternate meaning. For example, *workspace* in the above example has two different meanings, one in which it describes the contents of *image,* and one a spatial area which can itself be displayed. MURPHY cannot handle this second type of input since it cannot disambiguate multiple word senses. However, current research is directed toward including the ability to handle arbitrary numbers of word senses, as demonstrated by Dawson (1984), and future research must address this problem.

It is important to describe how MURPHY's ability to understand despite variant syntax derives from the integrated processing hypothesis. Syntactic knowledge is used by NLA during slot filling to help decide which CD on the C-LIST should fill a given slot. The syntactic features for a slot are grouped with the semantic preferences and semantic requirements for that slot, and candidate fillers are scored to see which features, preferences, and requirements each satisfies. The slot is filled with the candidate that (a) satisfies the most with the highest score and (b) is not already filling another slot with a higher score. This means that even though certain features or preferences may not be satisfied by a candidate, if that candidate nonetheless has the highest score it will be chosen as the filler. In particular, it means that a filler can be chosen even though some of the syntactic features that describe its intended position are false, as would be the case if the filler was out of position. In this case, the combination of the remaining true syntactic features, the semantic preferences, and the semantic restrictions supply enough information to determine the correct meaning of the utterance even though the word is out of position. Thus, the ability to understand input with variant syntax is a direct consequence of MURPHY's processing syntax and semantics simultaneously and of its processing syntax and semantics with the same mechanism.

6.2 UNDERSTANDING INPUT WITH MISSING WORDS

Incomplete input prevents NLA from combining the C-LIST elements into a single CD and causes the C-LIST elements to have unfilled slots. When this occurs, RBE completes the understanding process by inferring additional concepts. To understand this inference process, consider the following example, similar to (2) in the example on page 91.

> Threshold the current image at fifty.
DO YOU MEAN "threshold the current workspace image at fifty"?

> Yes.
OK
(The system displays the operation occurring.)

In this example, the user's utterance was missing the word *workspace.* NLA understands this utterance as well as possible, but fails to find a filler for the SOURCE slot in the meaning of *image.* RBE is called to infer a filler. First, the semantic requirements of the SOURCE slot are collected: the filler is required to be a possible image source. Then, RBE first searches the C-LIST for concepts whose attributes match this requirement. Not finding any on the C-LIST, RBE searches domain knowledge, and finds the meaning of *workspace.* This candidate is inserted into the empty slot, and the overall meaning is verified by the user.

The previous paragraph described the case in which the meanings of the missing words were slot fillers in frames supplied by other words contained in the utterance. What if one or more of those frame-supplying words were missing? For example, suppose the utterance did not contain the main concept word. In this case, NLA's partial understanding of the input will be in the form of several uncombined CDs. In this case, RBE must do more than merely fill empty slots. It must infer an entire additional CD which itself has empty slots that can be filled by the various CDs produced by NLA, as well as infer fillers for any remaining empty slots. This entire additional CD is inferred from context and domain knowledge. For example, consider interaction (6) from the example of section 2. In this interaction, MURPHY asked the user *What does a switch post look like?,* and the user responded *A metal cylinder two inches long.* The user's response is thus missing several words, including the main frame-building word *is.* NLA's best understanding is two isolated CDs representing the meaning of the phrases *a metal cylinder,* and *two inches long.* In order to understand the utterance as a whole, RBE must infer the meaning of the word *is,* which has slots for the meanings of these phrases, and must also fill a third slot in the meaning of *is* from domain knowledge with the meaning of the phrase *a switch post.* Note specifically that it is the meaning of the word *is* that is the main concept of the utterance, yet *is* was missing from the utterance. MURPHY does infer the meaning of *is,* fills two of its three empty slots with the meanings of the phrases understood by NLA, and fills the third with the meaning of *A switch post.* The user verifies this understanding, and thus MURPHY has understood despite a missing frame-building word.

The final example of understanding utterances that are missing words is the case in which the inferred slot filler itself has empty slots. For example, suppose the user's input is merely the utterance *Display,* as in the following example:

> Display.
DO YOU MEAN "Display the current workspace image"?
> Yes.
OK
(The system displays the image.)

Now, the meaning of *display* has one empty slot, that which is to be displayed, and MURPHY must infer this filler. However, suppose that the inferred filler is the meaning of *image.* In this case, there are additional slots to fill, namely the REF, TIME, and SOURCE slots, before the utterance will be completely understood. In order to understand the utterance *Display,* NLA first understands as well as possible. This results merely in the meaning of *display,* which is passed to RBE. RBE then searches context and domain knowledge for fillers of the IMAGE slot. It finds the meaning of *image,* which itself has empty REF, TIME, and SOURCE slots. RBE searches context and domain knowledge for fillers for these slots, and finds the meanings of *the, current,* and *workspace.* The user verifies these inferences, and understanding has been successful.

There is a second case, however, in which the fillers of the to-be-inferred CD have been supplied in the utterance. For example, consider the following:

> Display the current workspace.
DO YOU MEAN "Display the current workspace image"?
> Yes.
OK
(The system displays the image.)

In the user's input, the meaning of *display* has an empty IMAGE slot, which is to be filled by the inferred meaning of *image,* as before. The meaning of *image,* in turn, has empty REF, TIME, and SOURCE slots. Since RBE checks C-LIST elements before checking domain memory, it finds the fillers for these slots in the meanings of the input words *current* and *workspace* without searching domain knowledge.

How does this performance derive from the integrated processing hypothesis? Consider the case in which the utterance is missing one or more words. In this case, NLA will give RBE a CD containing one or more empty slots to be filled. RBE searches context and domain knowledge for appropriate fillers for each empty slot, and for empty slots in those fillers, and so on, until it builds a CD with no empty slots. If the user confirms the CD, RBE is done. If not, RBE resumes searching to infer the next most likely set of fillers. Since RBE evaluates the suitability of fillers for slots in the same way NLA does, it is the fact that language processing is fundamentally the same as memory processing that allows MURPHY to understand despite missing words. Moreover, this identity allows MURPHY to understand utterances in which the missing word's meaning contains a slot that must be filled by the meaning of a word that *was* present in the input. Since, presumably, meanings derived from input words should be used before meanings not so derived, RBE merely searches the C-LIST before searching context and domain knowledge, and gives preference to candidates found there. Thus processing language and memory using the same mechanism enables MURPHY to infer the meanings of missing words yet give preference to meanings derived from input words.

6.3 UNDERSTANDING ELLIPSES

Understanding ellipses requires that knowledge of the conversation be established, from which MURPHY can infer what was omitted. MURPHY maintains a conversational history by decomposing the meaning of each input into its component parts and appending them to domain knowledge. Thus conversational history is the initial part of domain knowledge, with the meanings of the most recent inputs first. Thus when RBE searches domain knowledge to complete understanding of an incomplete input, it first finds concepts derived from the meanings of the most recent user inputs. If any of these concepts are appropriate for understanding, then they will be part of the most likely interpretation of the input. Consider the following example:

> Where is the switch contact?
THE SWITCH CONTACT IS NEXT TO THE SWITCH BASE
> The switch post?
DO YOU MEAN "Where is the switch post?"
> Yes.

After MURPHY answers the user's first question, it adds the meanings of *where* and *is* to conversational history (as well as the rest of the component meanings of the question). When the user subsequently asks *The switch post?,* NLA first understands this phrase in isolation. The resulting meaning is not a goal concept, so RBE retrieves goal concepts from conversational history, finds the meaning of *is* previously placed there, and fills its OBJECT slot with the meaning of *The switch post.* It then examines the result to find additional empty slots, finds the VAL slot, and fills it with the meaning of *where* also retrieved from conversational history. MURPHY understood an elliptical input, and has used its conversational history to do so more rapidly than it would have otherwise.

MURPHY's ability to understand ellipses derives from the fact that it processes language and memory using the same mechanism. When the user's input is elliptical, NLA's understanding is incomplete. In order to fill empty slots and combine uncombined CDs, RBE searches domain knowledge. Since the first part of domain knowledge contains conversational history, RBE will search conversational history as it attempts to complete the understanding. Thus, if the utterance is elliptic, the appropriate concepts from conversational history will be found by RBE during its normal search. Since this search is the same process as that used by NLA and by RBE as it searches domain knowledge proper, MURPHY's ability to understand elliptic utterances derives from the fact that it processes language and memory using the same mechanism.

6.4 CORRECTABILITY

MURPHY's ability to enable the user to correct incorrect inferences operates very similarly to its ability to understand elliptical utterances. This ability is needed when MURPHY performs inferences in response to an incomplete input. Instead of merely saying "no" to MURPHY when it asks if its interpretation is correct, the user can provide a correction. MURPHY invokes NLA to understand the correction, and then adds its understood meaning to domain knowledge as if it were conversational history. Then, when RBE is searching further to infer the next possible interpretation for the user's original utterance, the first possibility found will be the meaning of the correction. Assuming the correction was indeed correct, RBE's next inference will contain the correction, the user will verify it, and understanding will be complete.

For example, suppose the user intended to type *Where is the switch contact?* but omitted the word *contact* and typed instead *Where is the switch?* Since the switch has not been assembled yet, NLA does not understand this utterance as referring to the switch as a whole. Rather, it understands only that the user is asking for the location of something, and does not know what role the meaning of *switch* plays in the utterance. Therefore RBE must infer the object whose location is being asked, and must try to use the meaning of *switch* to fill a slot in that inferred object's CD. It happens that in this case it would first find the meaning of *post,* then of *base,* and finally of *contact,* and use the meaning of *switch* to fill the PARTOF slot in each:

> Where is the switch?
DO YOU MEAN "Where is the switch post"?
> No
DO YOU MEAN "Where is the switch base"?
> No
DO YOU MEAN "Where is the switch contact"?
> Yes.
THE SWITCH CONTACT IS NEXT TO THE SWITCH BASE.

Thus MURPHY eventually understands, but conjectures two incorrect meanings before conjecturing the correct one. However, if the user offers a correction, RBE's search is concluded more quickly, as shown below.

> Where is the switch?
DO YOU MEAN "Where is the switch post"?
> No, contact.
THE SWITCH CONTACT IS NEXT TO THE SWITCH BASE.

MURPHY's ability to generate a series of possible input meanings ordered by likelihood is a consequence of the fact that the processing performed by NLA and RBE is a best-first search through a tree of semantic structures. Leaves of the tree are possible meanings of the input; since processing is best-first, MURPHY produces the most likely meaning first. However, if the first meaning is incorrect, the search can be resumed and the next most likely meaning obtained. Further, the fact that the appropriate data structures are maintained during search allows the understood meanings of corrections to be inserted when available, and yields a biased search in favor of the corrections if the first conjecture was incorrect. Since MURPHY's ability to perform best-first understanding results from the fact that NLA and RBE cooperate to perform best-first search, and since this is a consequence of the fact that both implement the same algorithm, MURPHY's ability stems from its use of the same mechanism for both language and memory processing, and hence from the integrated processing hypothesis.

6.5 OVERALL PERFORMANCE

MURPHY can be evaluated with respect to two further measures of performance. The first is the number of incorrect understandings MURPHY must generate before inferring the intended meaning. This depends on the number of concepts that must be inferred to complete understanding, the number of candidate fillers for those slots that exist in domain knowledge, and the number of alternative assignments of fillers to slots within those meanings deriving from the input words. The product of these parameters is the number of possible meanings of a given input, and the order in which these are generated is a function of the order of concepts in conversational history and domain knowledge and an ordering imposed by semantic preferences during search. Since the intended meaning is one of the possible meanings and hence has a position in this order, the number of incorrect meanings which must be generated are those that lie before the correct meaning in this order. In practice MURPHY usually generates from zero to three incorrect meanings before inferring the correct meaning; generally the constraints imposed by the knowledge representation renders tractable the combinatorially explosive number of possibilities. (Scaling the approach to richer environments will require much more sophisticated models of context – not the focus of the research reported here.) If a correction is given, of course, then the very next meaning is usually correct.

The second measure of performance is the question of whether MURPHY really understands all cases of missing words, ellipses, and out-of-order words. Although MURPHY has not been tested on every possible input, its understanding mechanisms implement a best-first search mechanism (Nilsson 1971) that exhaustively searches the space of possible meanings of an input utterance in an order determined by the meanings of input words, conversational history, and domain knowledge as described earlier. Thus, since "understand" means "eventually understand" in this paper, and since the search is exhaustive, MURPHY will understand any input, including all cases of missing words, ellipses, and out-of-order words, as long as that meaning is representable within the CDs MURPHY knows about.Indeed, MURPHY will attempt to make sense of deliberate nonsense, and if it can do so using the concepts in the input, conversa-

tional history and domain knowledge, and if the user confirms its interpretation at some point, it will succeed. Given this processing strategy, however, what prevents MURPHY from generating almost any conceivable CD in its attempt to correct errors, and thus generating wildly unreasonable guesses? In fact, wildly unreasonable guesses are not ruled out at all; MURPHY must be able to generate such guesses in case they represent the intended meaning. However, since MURPHY generates possible understandings according to preference, it will only generate wildly unreasonable guesses if the user has failed to verify all of the more reasonable guesses, in which case such a guess has actually become the most reasonable remaining.

6.6 ROBUST UNDERSTANDING

This section has described MURPHY's performance on inputs with variant syntax, missing words, ellipsis, with and without corrections. It has further considered its response time, the number of incorrect guesses it makes, and the degree to which it eventually understands all cases of such inputs. It appears that MURPHY displays the characteristics of robustness described in section 2.

7 INTEGRATED PROCESSING PRODUCES ROBUST UNDERSTANDING

This paper has described research based on the conjecture that, since the integrated processing hypothesis is a model of how people understand language, and since people are known to understand robustly, then a natural language interface incorporating the integrated processing hypothesis should prove robust. It appears that this approach has validity, since MURPHY is indeed robust, and this robustness derives from its embodiment of the integrated processing hypothesis. Specifically, MURPHY appears to be robust in the areas described in section 2. It successfully understands utterances that are missing words and have variant syntax, both without and with corrections. This performance thus supports the IPPRU conjecture.

However, MURPHY is not fully robust in the broadest sense. There remain other characteristics of real-world input which have not been considered here, such as false starts, unknown words, irrelevant interjections, and learning new words. In fact, however, MURPHY can already understand input with some of these characteristics, and extensions to those remaining appear possible. Selfridge (1980, 1981a, 1981b, 1982) describes the CHILD program which models child language learning. CHILD readily understands utterances that contain unknown words, and it readily learns new word meaning and syntax. Since MURPHY and CHILD use the same understanding and inference programs, MURPHY can also understand input with unknown words and can learn new word meanings and syntax. Understanding false starts and irrelevant interjections are really the same problem, and it appears that MURPHY can easily understand despite them. To see this, recall the basic slot-filling mechanism MURPHY uses, and consider how the search for candidate fillers is carried out: those CDs that fail to satisfy the semantic requirements are not considered further. Since in all likelihood the meaning of false starts and interjections will fail to satisfy any requirements, most of the time they will be ignored and understanding will proceed as if they were not present. In those rare cases in which the meaning of a false start or interjection can incorrectly, but plausibly, be incorporated into the understood meaning of the utterance, the user will fail to verify it and MURPHY will generate another, eventually correct, understanding. MURPHY's robustness thus extends considerably beyond that reported in this paper, and it appears to represent a promising approach for future work.

Such future work will concentrate in several specific areas. First, of course, is the complete unification of NLA and RBE, including as much interaction as possible with CCON. This represents the step from merely embodying the integrated processing hypothesis to actually being integrated. In addition, the ability to handle multiple word senses is critical; Dawson (1984) has extended the preference algorithms described here to handle multiple word senses, and it remains only to incorporate them in MURPHY. Further, MURPHY's representation of syntax is probably inadequate for utterances of significantly greater complexity than those currently handled. This representation will either have to be extended, or a new representation developed. Its representation of semantics is similarly weak. The technique of characterizing concepts by whether they satisfy semantic requirements and preferences is too simple, and should be extended to include complex reasoning about concepts. Finally, MURPHY's search algorithms require improvement. While they will certainly work in realistically large domains, they will probably prove unreasonably slow to infer the intended meaning. Indeed, while human language processing is integrated (Schank and Birnbaum 1981), humans are not capable of generating a complete sequence of possible meanings in order of likelihood (Kolodner 1980). Rather, humans employ high-level reasoning and inference within a rich and highly structured memory, and usually infer the intended meaning quickly. It would be desirable to integrate MURPHY with a system employing such a memory (Dyer 1982, Lebowitz 1980) in order to improve its understanding and inference mechanisms.

Since MURPHY is robust in the desired ways, and its limitations appear clear, it appears that further explorations of the role of the integrated processing hypothesis in robust natural language interfaces should prove fruitful.

AKNOWLEDGEMENT

Roger Schank originated the paradigm within which the research presented here took place; thanks to Larry

Birnbaum for extensive discussions of the issues of integrated processing and for critically reading an earlier draft; thanks to Howard Blair for critically reading an earlier draft.

References

Birnbaum, L. and Selfridge, M. 1981 Conceptual Analysis. In: Schank, R. and Riesbeck, C.K., Eds., *Inside Computer Understanding: Five Programs plus Miniatures.* Lawrence Erlbaum Associates, Hillsdale, New Jersey, USA: 318-353.

Bobrow, R.J. and Webber, B.L. 1980 Knowledge Representation for Syntactic/Semantic Processing. *Proceedings of the First Annual Conference of the AAAI.* Stanford University, California: 316-323.

Carbonell, J.G. and Hayes, P.J. 1983 Recovery Strategies for Parsing Extragrammatical Language. *American Journal of Computational Linguistics* 9(3-4): 123-146

Cullingford, R. 1978 Script Application: Computer Understanding Of Newspaper Stories. Research Report No. 116, Department of Computer Science, Yale University, New Haven, Connecticut.

Cullingford, R.; Krueger, M.; Selfridge, M.; and Bienkowski, M. 1981 Automated Explanations as a Component of a CAD System. *IEEE Transactions on Systems, Man and Cybernetics.* SMC-12.

Dawson, B. 1984 Preference Analysis with Arbitrary Numbers of Word Senses. MS Thesis, Department of Electrical Engineering and Computer Science, University of Connecticut, Storrs, Connecticut.

Dyer, M. 1982 In-Depth Understanding: A Computer Model Of Integrated Processing For Narrative Comprehension. Research Report No. 219, Department of Computer Science, Yale University, New Haven, Connecticut.

Engelberg, J. 1983 Integrated Processing Produces Robust Understanding. MS Thesis, Department of Electrical Engineering and Computer Science, University of Connecticut, Storrs, Connecticut.

Engelberg, J.; Levas, A.; and Selfridge, M. 1984 A Natural Language Interface to a Robot Assembly System. *Proceedings of the IEEE International Conference on Robotics.* Atlanta, Georgia: 400-403.

Fass, D. and Wilks, Y. 1983 Preference Semantics, Ill-formedness, and Metaphor. *American Journal of Computational Linguistics.* 9(3-4): 178-187.

Granger, R.H. 1984 The NOMAD System: Expectation-Based Detection and Correction of Errors During Understanding of Syntactically and Semantically Ill-formed Text. *American Journal of Computational Linguistics* 9(3-4): 188-198.

Hayes, P.J. and Carbonell, J.G. 1981 Multi-Strategy Parsing and its Role in Robust Machine Communications. CMU-CS-81-118, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

Hayes, P.J. and Mouradian, G.V. 1981 Flexible Parsing. *American Journal of Computational Linguistics* 7(4): 232-242.

Kolodner, J. 1980 Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model. Research Report No. 187, Yale University, Dept. of Computer Science, New Haven, Connecticut.

Kwasny, S.C. and Sondheimer, N.K. 1981 Relaxation Techniques for Parsing Grammatically Ill-Formed Input in Natural Language Understanding Systems. *American Journal of Computational Linguistics,* 7(2): 99-108.

Lebowitz, M. 1980 Generalization and Memory in an Integrated Understanding System. Research Report No. 186, Department of Computer Science, Yale University, New Haven, Connecticut.

Levas, A. 1983 Teaching Robots Assembly Plans By Example. MS Thesis, Department of Electrical Engineering and Computer Science, University of Connecticut, Storrs, Connecticut.

Levas, A. and Selfridge, M. 1984 A User-Friendly High-level Robot Teaching System. *Proceedings of the IEEE International Conference on Robotics.* Atlanta, Georgia: 413-416.

Nilsson, Nils. 1971 *Problem-Solving Methods in Artificial Intelligence.* McGraw-Hill, New York.

Riesbeck, C. and Schank, R. 1976 Comprehension by Computer: Expectation-Based Analysis of Sentences in Context. Research Report No. 78, Department of Computer Science, Yale University, New Haven, Connecticut.

Schank, R. 1975 *Conceptual Information Processing.* Elsevier-North Holland, New York.

Schank, R. and Abelson, R. 1977 *Scripts, Plans, Goals and Understanding: An Inquiry into Human Knowledge Structures.* Lawrence Erlbaum Assoc., Hillsdale, New Jersey.

Schank, R. and Birnbaum, L. 1981 Memory, Meaning, and Syntax. Research Report No. 189, Department of Computer Science, Yale University, New Haven, Connecticut.

Selfridge, M. 1980 A Process Model of Language Acquisition. Ph.D Dissertation, Research Report No. 172, Department of Computer Science, Yale University, New Haven, Connecticut.

Selfridge, M. 1981a Why Do Children Say "Goed"? A Computer Model of Child Generation. *Proceedings of the Third Annual Meeting of the Cognitive Science Society.* Berkeley, California: 131-133.

Selfridge, M. 1981b A Computer Model of Language Acquisition. *Proceedings of the 7th International Joint Conference on Artificial Intelligence,* Vancouver, British Columbia: 92-96.

Selfridge, M. 1982 Why Do Children Misunderstand Reversible Passives? The CHILD Program Learns to Understand Passive Sentences. *Proceedings of the Third Annual Conference of the AAAI.* Pittsburgh, Pennsylvania: 251-254.

Selfridge, M. 1983 Natural Language Interfaces to Image Analysis Systems. *Proceedings of Trends and Applications, 1983.* Silver Spring, Maryland: 248-251.

Weischedel, R.M. and Sondheimer, N.K. 1983 Meta-rules as a Basis for Processing Ill-Formed Output. *American Journal of Computational Linguistics.* 9(3-4): 161-177

Wilks, Y. 1976 Parsing English II. In Wilks, Y. and Charniak, E., Eds., *Computational Semantics.* North-Holland Publishing Co., New York, New York: 155-184.

Woods, W.A. 1970 Transition Network Grammars for Natural Language Analysis. *Communications of the ACM* 13(10): 591-606.