Recovery Strategies for Parsing Extragrammatical Language[1]

Jaime G. Carbonell and Philip J. Hayes

Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA 15213

Practical natural language interfaces must exhibit robust behaviour in the presence of extragrammatical user input. This paper classifies different types of grammatical deviations and related phenomena at the lexical, sentential and dialogue levels and presents recovery strategies tailored to specific phenomena in the classification. Such strategies constitute a tool chest of computationally tractable methods for coping with extragrammaticality in restricted domain natural language. Some of the strategies have been tested and proven viable in existing parsers.

1. Introduction

Any robust natural language interface must be capable of processing input utterances that deviate from its grammatical and semantic expectations. Many researchers have made this observation and have taken initial steps towards coverage of certain classes of extragrammatical constructions. Since robust parsers must deal primarily with input that does meet their expectations, the various efforts at coping with extragrammaticality have been generally structured as extensions to existing parsing methods. Probably the most popular approach has been to extend syntactically-oriented parsing techniques employing Augmented Transition Networks (ATNs) (Kwasny and Sondheimer 1981, Weischedel and Sondheimer 1984, Weischedel and Black 1980, Woods et al. 1976). Other researchers have attempted to deal with ungrammatical input through network-based semantic grammar techniques (Hendrix 1977), through extensions to pattern matching parsing in which partial pattern matching is allowed (Hayes and Mouradian 1981), through conceptual case frame instantiation (Dejong 1979, Schank, Lebowitz, and Birnbaum 1980), and through approaches involving multiple cooperating parsing strategies (Carbonell and Hayes 1984, Carbonell et al. 1983, Hayes and Carbonell 1981).

Given the background of existing work, this paper focuses on three major objectives:

1. to create a taxonomy of possible grammatical deviations covering a broad range of extragrammaticalities, including some lexical and discourse phenomena (for example, novel words and dialogue level ellipsis) that can be handled by the same mechanisms that detect and process true grammatical errors;
2. to outline strategies for processing many of these deviations – some of these strategies have been presented in our earlier work, some are similar to strategies proposed by other researchers, and some have never been analyzed before;
3. to assess how easily these strategies can be employed in conjunction with several of the existing approaches to parsing ungrammatical input, and to examine why mismatches arise.

The overall result should be a synthesis of different parse-recovery strategies organized by the grammatical phenomena they address (or violate), an evaluation of how well the strategies integrate with existing approaches to parsing extragrammatical input, and a set of characteristics desirable in any parsing process dealing with extragrammatical input. We hope this will aid researchers designing robust natural language interfaces in two ways:

1. by providing a tool chest of computationally effective approaches to cope with extragrammaticality;

[1] This research was sponsored in part by the Air Force Office of Scientific Research under Contract AFOSR-82-0219 and in part by Digital Equipment Corporation as part of the XCALIBUR project.

Copyright 1984 by the Association for Computational Linguistics. Permission to copy without fee all or part of this material is granted provided that the copies are not made for direct commercial advantage and the *Journal* reference and this copyright notice are included on the first page. To copy otherwise, or to republish, requires a fee and/or specific permission.

0362-613X/83/030123-24$03.00

2. by assisting in the selection of a basic parsing methodology in which to embed these recovery techniques.

In assessing the degree of compatibility between recovery techniques and various approaches to parsing, we avoid the issue of whether a given recovery technique ***can*** be used with a specific approach to parsing. The answer to such a question is almost always affirmative. Instead, we are concerned with how ***naturally*** the recovery strategies fit with the various parsing approaches. In particular, we consider the computational tractability of the recovery strategies and how easily they can obtain the information they need to operate in the context of different parsing approaches.

The need for robust parsing is greatest for interactive natural language interfaces that have to cope with language produced spontaneously by their users. Such interfaces typically operate in the context of a well-defined, but restricted, domain in which strong semantic constraints are available. In contrast, text processing often operates in domains that are semantically much more open-ended. However, the need to deal with extragrammaticality is much less pronounced in text processing, since texts are normally carefully prepared and edited, eliminating most grammatical errors and suppressing many dialogue phenomena that produce fragmentary utterances. Consequently, we shall emphasize recovery techniques that exploit and depend on strong semantic constraints. In some cases, it is unclear whether the techniques we discuss will scale up properly to unrestricted text or discourse, but even where they may not, we anticipate that their use in the restricted situation will provide insights into the more general problem.

Before proceeding with our discussion, the term *extragrammaticality* requires clarification. *Extragrammaticalities* include patently ungrammatical constructions, which may nevertheless be semantically comprehensible, as well as lexical difficulties (for example, misspellings), violations of semantic constraints, utterances that may be grammatically acceptable but are beyond the syntactic coverage of the system, ellipsed fragments and other dialogue phenomena, and any other difficulties that may arise in parsing individual utterances. An *extragrammaticality* is thus defined with respect to the capabilities of a particular system, rather than with respect to an absolute external competence model of the ideal speaker.

Extragrammaticality may arise at various levels: lexical, sentential, and dialogue. The following sections examine each of these levels in turn, classifying the extragrammaticalities that can occur, and discussing recovery strategies. At the end of each section, we consider how well the various recovery strategies would fit with or be supported by various approaches to parsing. A final section discusses some experimental robust parsers that we have implemented. Our experience with these parsers forms the basis for many of the observations we offer throughout the paper. We also discuss some more recent work on integrating many of the recovery strategies considered earlier into a single robust multi-strategy parser for restricted domain natural language interpretation.

2. Lexical Level Extragrammaticalities

One of the most frequent parsing problems is finding an unrecognizable word in the input stream. The following subsections discuss the underlying reasons for the presence of unrecognizable words and develop applicable recovery strategies.

2.1. The unknown word problem

The word is a legitimate lexeme but is not in the system's dictionary. There are three reasons for this:

- The word is outside the intended coverage of the interface (For example, there is no reason why a natural language interface to an electronic mail system should know words like "chair" or "sky", which cannot be defined in terms of concepts in its semantic domain).
- The word refers to a legitimate domain concept or combination of domain concepts, but was not included in the dictionary. (For example, a word like "forward" [a message] can be defined as a command verb, its action can be clearly specified, and the objects upon which it operates – an old message and a new recipient – are already well-formed domain concepts.)
- The word is a proper name or a unique identifier, such as a catalogue part name/number, not heretofore encountered by the system, but recognizable by a combination of contextual expectations and morphological or orthographic features (for example, capitalization).

In the first situation, there is no meaningful recovery strategy other than focused interaction (Hayes 1981) to inform the user of the precise difficulty. In the third, little action is required beyond recognizing the proper name and recording it appropriately for future reference. The second situation is more complicated; three basic recovery strategies are possible:

1. Follow the KLAUS (Haas and Hendrix 1983) approach, where the system temporarily wrests initiative from the user and plays a well designed "twenty questions" game, classifying the unknown term syntactically, and relating it semantically to existing concepts encoded in an inheritance hierarchy. This method has proven successful for verbs, nouns and adjectives, but only when they turn out to be instances of predefined general classes of objects and actions in the domain model.
2. Apply the *project and integrate* method (Carbonell 1979) to infer the meaning and syntactic category of the word from context. This method has proven

useful for nouns and adjectives whose meaning can be viewed as a recombination of features present elsewhere in the input. Unlike the KLAUS method, it operates in the background, placing no major run-time burden on the user. However, it remains highly experimental and may not prove practical without user confirmation.

3. Interact with the user in a focused manner to provide a paraphrase of the segment of input containing the unknown word. If this paraphrase results in the desired action, it is stored and becomes the meaning of the new word in the immediate context in which it appeared. The LIFER system (Hendrix 1977) had a rudimentary capacity for defining synonymous phrases. A more general method would generalize synonyms to classify the new word or phrase in different semantic contexts.

2.2.2 Misspellings

Misspellings arise when an otherwise recognizable lexeme has letters omitted, substituted, transposed, or spuriously inserted. Misspellings are the most common form of extragrammaticality encountered by natural language interfaces. Usually, a word is misspelt into an unrecognizable character string. But, occasionally a word is misspelt into another word in the dictionary that violates semantic or syntactic expectations. For instance:

> "Copy the flies from the accounts directory to my directory"

Although "flies" may be a legitimate word in the domain of a particular interface (for example, the files could consist of statistics on med-fly infestation in California), it is obvious to the human reader that there is a misspelling in the sentence above.

There are well-known algorithms for matching a misspelt word against a set of possible corrections (Durham, Lamb, and Saxe 1983), and the simplest recovery strategy is to match unknown words against the set of all words in an interface's dictionary. However, this obviously produces incorrect results when a word is misspelt into a word already in the dictionary, and can produce unnecessary ambiguities in other cases.

Superior results are obtained by making the spelling correction sensitive to the parser's syntactic and semantic expectations. In the following example:

> Add two fixed haed dual prot disks to the order

"haed" can be corrected to: "had", "head", "hand", "heed", and "hated". Syntactic expectations rule two of these out, and domain semantics rule out two others, leaving "fixed head disk" as the appropriate correction. Computationally, there are two ways to organize this. One can either match parser expectations against all possible corrections in the parser's current vocabulary, and rule out spurious corrections, or one can use the parse expectations to generate a set of possible words that can be recognized at the present point and use this as input to the spelling correction algorithm. The latter, when it can be done, is clearly the preferable choice on efficiency criteria. Generating all possible corrections with a 10,000 word dictionary, only to rule out all but one or two, is a computationally-intensive process, whereas exploiting fully-indexed parser expectations is far more constrained and less likely to generate ambiguity. For the example above, "prot" has 16 possible corrections in a small on-line dictionary. However, domain semantics allow only one word in the same position as "prot", so correction is most effective if the list of possible words is generated first.

2.3. Interaction of morphology and misspelling

Troublesome side-effects of spelling correction can arise with parsers that have an initial morphological analysis phase to reduce words to their root form. For instance, a parser might just store the root form of 'directory' and reduce 'directories' to 'directory' plus a plural marker as part of its initial morphological phase. This process is normally driven by first failing to recognize the inflected form as a word that is present in the dictionary, and then applying standard morphological rules (for example, **–ies => +y**) to derive a root from the inflected form. If any root thus derived is in the dictionary, the input word is assumed to be the appropriate inflected form.

There are several ways in which this procedure can interact with spelling correction:

1. The same test, viz. not finding the word in the dictionary, is used to trigger both morphological analysis and spelling correction, so there is a question of which to do first.
2. The root of the word may be misspelt (e.g. dircto-ries), even though the inflexion is correct, so that after the inflexion is removed, there is still no matching dictionary entry.
3. The inflexion itself may be misspelt (e.g. director-ise), so that the standard morphological transformations do not apply.

The first kind of interaction is not usually a major problem. On the assumption that inflexion is more common than misspelling, the most straightforward and probably best strategy is to try inflexion first on unknown words and then if that does not produce a word in the dictionary, try spelling correction. Matching only against contextually appropriate words should avoid cases in which a misspelling produces an inflected form of a different word.

If the root of an inflected word is misspelt, it will be necessary to spelling correct all of the (possibly several) uninflected forms, which might be inefficient. Again, contextual sensitivity can help.

The third kind of interaction is most troublesome. Most inflexions are too short for spelling correction to be effective – letter substitution or omission on two letter sequences is hard to identify. Moreover, inflexion processing does not normally use an explicit list of inflexions, but instead is organized as a discrimination net, containing the inflexions implicitly. One solution may be to have a list of all misspellings of inflected forms, but even utilizing hash coding schemes, searching this set would be inefficient.

A simpler solution to the entire problem of interaction between spelling correction and morphological analysis is to eliminate the morphological analysis, and just store all inflected forms in the dictionary. This has the disadvantages of being unaesthetic and being unable to deal with novel inflexions, but neither of these are major problems for restricted domain natural language interfaces. There is also a second order problem in that more than one inflected form of the same word could be found as candidate corrections through spelling correction, but this can be overcome by explicitly grouping the various inflexions of a given root together in the lexicon.

2.4. Incorrect segmentation

Input typed to a natural language interface is segmented into words by spaces and punctuation marks. Both kinds of segmenting markers, especially the second, can be omitted or inserted speciously. Incorrect segmentation at the lexical level results in two or more words being run together, as in "runtogether", or a single word being split up into two or more segments, as in "tog ether" or (inconveniently) "to get her", or combinations of these effects as in "runto geth er".

In all these cases, it is possible to deal with such errors by extending the spelling correction mechanism to be able to recognize target words as initial segments of unknown words, and vice-versa. For instance, by spelling correcting "portdisks" against what is acceptable in the position it occupies in:

Add two dual portdisks to the order

it should be possible to recognize the initial segment "port" as the intended word, with "disks" as a left over segment to be inserted into the input string after the corrected word for further parsing, resulting in this case in the correct parse. Again, in:

Add two dual port disks to the ord er

an unrecognized (and uncorrectable) word "er" following a word "ord" which has been recognized as an initial segment abbreviation should trigger an attempt to attach the unknown word to the end of the abbreviation to see if it completes it. Correction of

Add two du alport disks to the order

would be somewhat harder. After failing in the above recovery methods, one letter at a time would be stripped off the beginning of the second unrecognizable word ("alport") and added at the end of the first unrecognizable word ("du"). This process succeeds only if at some step both words are recognizable and enable the parse to continue. Migrating the delimiter (the space) backwards as well as forwards should also be attempted between a pair of unknown words, stopping if both words become recognizable. Of course, the compounding of multiple lexical deviations (for example, misspellings, run-on words and split words in the same segment) requires combinatorially inefficient recovery strategies. Strong parser expectations ameliorate this problem partially, but trade-offs must be made between resilience and efficiency for compound error recovery.

2.5. Support for recovery strategies by various parsing approaches

In general, lexical level recovery strategies operate in a sufficiently localized manner that the variations in global behaviour of different approaches to parsing do not come into play. However, most of the strategies are capable of using contextual restrictions on what incorrect lexical item might be, and therefore are most effective when the constraints on the unknown word are strongest. This suggests that they will be most successful when used with an approach to parsing in which it is easy to bring semantic constraints to bear. So, for instance, such techniques are likely to be more effective using a semantic grammar (Hendrix 1977, Brown and Burton 1975) or case frame instantiation (Dejong 1979, Hayes and Carbonell 1981) approach, than in an approach using a syntactic ATN (Woods, Kaplan and Nash-Webber 1972), where the expectations are never more specific than membership in one or more general syntactic categories.

3. Sentential Level Extragrammaticalities

Recovery from extragrammaticality at the sentential level is much more dependent on the particular kind of parsing techniques that are employed. Some techniques lend themselves to straightforward recovery methods, while others make recovery difficult. An initial examination of the requirements for recovery from various kinds of sentential level ungrammaticality will allow us to draw some general conclusions about the most suitable basic parsing approaches to build on. We examine ungrammaticalities in five categories: missing words, spurious words or phrases, out of order constituents, agreement violations, and semantic constraint violations.

3.1. Missing constituents

It is not uncommon for the user of a natural language interface to omit words from his input, either by mis-

take or in an attempt to be cryptic. The degree of recovery possible from such ungrammaticalities is, of course, dependent on which words were left out. For instance in:

> Add two fixed head dual ported disks to my order

omitting "dual" would be unrecoverable since all disks are ported and the discriminating information about the number of ports would not be there. On the other hand, if "ported" is omitted, all vital information is still there (the only thing dual about disks is the number of ports) and it should be possible to recover. Also the omission of function words like prepositions or determiners is usually (though not always) recoverable. In practice, most omissions are of words whose contribution to the sentence is redundant, and are done consciously in an attempt to be cryptic or "computer-like" (as in "Copy new files my directory"). This suggests that techniques that fill in the gaps on semantic grounds are more likely to be successful than strategies that do not facilitate the application of domain semantics.

In general, coping with missing words requires a parsing process to determine the parse structure that would have been obtained if those words had been there. If the information provided by the missing words is not redundant (as in the case of "dual" above), then this structure will have gaps, but the structure will convey the broad sense of the user's intention, and the gaps can be filled in by inference or (more practically and safely) by interaction with the user, focusing on the precise gaps in the context of the global parse structure (see Section 4.2 for further discussion of focused interaction techniques.)

A parsing process postulates a missing word error when its expectations (syntactic or semantic) of what should go at a certain place in the input utterance are violated. To discover that the problem is in fact a missing word, and to find the parse structure corresponding to the user's intention, the parsing process must "step back" and examine the context of the parse as a whole. It needs to ignore temporarily the unfulfilled expectations and their contribution to the overall structure while it tries to fulfil some of its other expectations through parsing other parts of the input and integrating them with already parsed constituents. In terms of a left-to-right parse of the above example (minus "dual"), this would mean that when the parser encountered "ported", it should note that even though it was expecting the start of a modifier suitable for a computer component (assuming its expectations are semantic), it had in fact found the latter part of a modifier for a disk and so could proceed as though the whole of the modifier was there. A parser with greater directional freedom might find "disk" first and then look more specifically for qualifiers suitable for disks. Again, the existence of a complete disk qualifier in the user's intended utterance could be assumed from finding part of the qualifier in a place where a whole one should go.

Another way of looking at this is as an attempt to delimit the gap in the input utterance, correlate it with a gap in the parse structure (filling in that gap if it is uniquely determined), and realign the parsing mechanism as though the gap did not exist. Such a realignment can be done top-down by hypothesizing the other expected constituents from the parse structure already obtained and attempting to find them in the input stream. Alternatively, realignment can be done bottom-up by recognizing as yet unparsed elements of the input, and either fitting them into an existing parse structure, or finding a larger structure to subsume both them and the existing structure. This latter approach is essential when the structuring words are missing or garbled.

Whether a top-down or a bottom-up method is best in any given instance will depend on how much structure the parser can recognize before having to deal with the missing word. If the parser is left-to-right and the gap appears early in the input, there is likely to be little structure already built up, so a bottom-up approach will probably produce better results. Similarly, if the missing word itself provides the highest level of structure (for example, "add" in the example above), a bottom-up approach is essential. On the other hand, if the missing word corresponds to a spot low-down in the parse structure, and the gap is late in the utterance, or the parser is not bound to a strict left-to-right directionality, a top-down approach is likely to be much more efficient. In general, both methods should be available.

3.2. Spurious constituents

Words in an input utterance that are spurious to a parse can arise from a variety of sources:

- **legitimate phrases that the parser cannot deal with:** It is not uncommon for the user of a restricted domain interface to say things that the interface cannot understand because of either conceptual or grammatical limitations. Sometimes, spurious verbosity or politeness is involved:

  > Add if you would be so kind two fixed head and if possible dual ported disks to my order.

  Or the user may offer irrelevant (to the system) explanations or justifications, as observed in preparatory experiments for the GUS system (Bobrow et al. 1977), for example,

  > I think I need more storage capacity, so add two fixed head dual ported disks to my order.

Some common phrases of politeness can be recognized explicitly, but in most cases, the only reasonable response is to ignore the unknown phrases, realign the parse on the recognizable input, and if a semantically and syntactically complete structure results, postulate that the ignored segment was indeed redundant. In most such cases, the user should be informed that part of the input was ignored.

- **broken-off and restarted utterances:** These occur when people start to say one thing, change their mind, and say another:

  Add I mean remove a disk from my order

  Utterances in this form are more likely to occur in spoken input, but a similar effect can arise in typed input when a user forgets to hit the erase line or erase character key:

  Add remove a disk from my order
  Add a single ported dual ported disk from my order

  Again the best tactic is to discard the broken-off fragment, but identifying and delineating the superseded fragment requires strategies such as the one discussed below.
- **unknown words filling a known grammatical role:** Sometimes the user will generate an incomprehensible phrase synonymous with a constituent the system is perfectly capable of understanding:

  Add a dual ported rotating mass storage device to my order

  Here the system might not know that "rotating mass storage device" is synonymous with "disk". This phenomenon will result in missing words as well as spurious words. If the system has a unique expectation for what should go in the gap, it should (with appropriate confirmation from the user) record the unknown words as synonymous with what it expected. If the system has a limited set of expectations for what might go in the gap, it could ask the user which one (if any) he meant and again record the synonym for future reference. In cases where there are no strong expectations, the system would ask for a paraphrase of the incomprehensible fragment. If this proved comprehensible, it would then postulate the synonymy relation, ask the user for confirmation, and again store the results for future reference.

The kind of recovery strategies required here are surprisingly similar to those required for missing words. Essentially, the parser must recognize that the input contains recognizable segments as well as unexpected and unrecognizable words and phrases interspersed among them. The way that a parser (at least a left-to-right parser) would encounter the problem is identical to the way that missing words are manifested, viz. the next word in sequence does not fulfil the parser's expectations. Overcoming this problem involves the same notion of "stepping back" and seeing how subsequent elements of the input fit with the parsing structure built up so far. A major difference is that the word that violated the expectations, and possibly other subsequent words, may not be incorporated into the resulting structure. Moreover, in the case of purely spurious phrases, that structure may not have any gaps. For a parser with more directional freedom, the process of finding spurious phrases may be simpler in that it could parse all the words that fit into the structure before concluding that the unrecognizable words and phrases were indeed spurious. When gaps in the parse structure remain after parsing all the recognizable input, the unrecognizable segment may not be spurious after all. It can be aligned with the gap in the parse and the possible synonymy relations discussed above can be presented to the user for approval.

In the case of broken-off utterances, there are some more specific methods that allow the spurious part of the input to be detected:

- If a sequence of two constituents of identical syntactic and semantic type is found where only one is permissible, simply ignore the first constituent. Two main command verbs in sequence (for example, as in "Add remove ..." above), instantiate the identical sentential case header role in a case frame parser, enabling the former to be ignored. Similarly, two instantiations of the same prenominal case for the "disk" case frame would be recognized as mutually incompatible and the former again ignored. Other parsing strategies can be extended to recognize equivalent constituent repetition, but case frame instantiation seems uniquely well suited to it.
- Recognize explicit corrective phrases and if the constituent to the right is of equivalent syntactic and semantic type as the constituent to the left, substitute the right constituent for the left constituent and continue the parse. This strategy recovers from utterances such as "Add I mean remove ...", if "I mean" is recognized as a corrective phrase.
- Select the minimal constituent for all substitutions. For instance in

  Add a high speed tape drive, that's disk drive, to the order

  one desires "disk drive" to substitute for "tape drive", not for the larger phrase "high speed tape drive", which also forms a legitimate constituent of like semantic and syntactic type. This preference is based solely on pragmatic grounds and empirical evidence.

In addition to identifying and ignoring spurious input, a robust interface must tell the user what it has

ignored and should paraphrase the part of the input that it did recognize. The unrecognized input may express vital information, and if that information is not captured by the paraphrase, the user may wish to try again. Exceptions to this rule arise when the spurious input can be recognized explicitly as such. Expressions of politeness, for instance, might be treated this way. The ability to recognize such "noise" phrases makes them in some sense part of the expectations of the parser, and thus not truly spurious. However, isolating them in the same way as spurious input provides the advantage that they can then be recognized at any point in the input without having to clutter the parser's normal processing with expectations about where they might occur.

3.3. Out of order constituents and fragmentary input

Sometimes, a user will use non-standard word order. There are a variety of reasons why users violate expected constituent ordering relations, including unwillingness to change what has already been typed, especially when extensive retyping would be required.

> Two fixed head dual ported disk drives add to the order

or a belief that a computer will understand a clipped pseudo-military style more easily than standard usage:

> two disk drives fixed head dual ported to my order add

Similar myths about what computers understand best can lead to a very fragmented and cryptic style in which all function words are eliminated:

> Add disk drive order

instead of "add a disk drive to my order".

These two phenomena, out of order constituents and fragmentary input, are grouped together because they are similar from the parsing point of view. The parser's problem in each case is to put together a group of recognizable sentence fragments without the normal syntactic glue of function words or position cues to indicate how the fragments should be combined. Since this syntactic information is not present, semantic considerations have to shoulder the burden alone. Hence, parsers that make it easy for semantic information to be brought to bear are at a considerable advantage.

Both bottom-up and top-down recovery strategies are possible for detecting and recovering from missing and spurious constituents. In the bottom-up approach, all the fragments are recognized independently, and purely semantic constraints are used to assemble them into a single framework meaningful in terms of the domain of discourse. When the domain is restricted enough, the semantic constraints can be such that they always produce a unique result. This characteristic was exploited to good effect in the PLANES system (Waltz 1978) in which an input utterance was recognized as a sequence of fragments which were then assembled into a meaningful whole on the basis of semantic considerations alone. A top-down approach to fragment recognition requires that the top-level or organizing concept in the utterance ("add" in the above examples) be located first and the predictions obtainable from it about what else might appear in the utterance be used to guide and constrain the recognition of the other fragments.

As a final point, note that in the case of out of order constituents, a parser relying on a strict left-to-right scan will have much greater difficulty than one with more directional freedom. In out of order input, there may be no meaningful set of left-to-right expectations, even allowing for gaps or extra constituents, that will fit the input. For instance, a case frame parser that scans for the head of a case frame, and subsequently attempts to instantiate the individual cases from surrounding input, is far more amenable to this type of recovery than one dependent upon rigid word order constraints.

3.4. Syntactic and semantic constraint violations

Input to a natural language system can violate both syntactic and semantic constraints. The most common form of syntactic constraint violation is agreement failure between subject and verb or determiner and head noun:

> Do the order include a disk drives?

Semantic constraint violations can occur because the user has conceptual problems:

> Add a floating head tape drive to the order

or because he is imprecise in his language, using a related object in place of the object he really means. For instance, if he is trying to decide on the amount of memory to include in an order he might say

> Can you connect a video disk drive to the two megabytes.

when what he really means is "... to the computer with two megabytes of memory".

These different kinds of constraint violation require quite different kinds of treatment. In general, the syntactic agreement violations can be ignored; cases in which agreement or lack of it distinguishes between two otherwise valid readings of an input are rare. However, one problem that sometimes arises is knowing whether a noun phrase is singular or plural when the determiner or quantifier disagrees with the head noun. It is typically best to let quantifiers dominate when they are used; for example, "two disk" really

means "two disks". And with determiner disagreement, it is often unimportant which reading is taken. In the example of disagreement above, it does not matter whether the user meant "a disk drive" or "any disk drives". The answer will be the same in either case, viz. a listing of all the disk drives that the order contains. In cases where the action of the system would be different depending on whether the noun phrase was singular or plural (e.g. "delete a disks from the order"), the system should interact with the user in a focused way to determine what he really meant.

Semantic constraint violations due to a user's conceptual problems are harder to deal with. Once detected, the only solution is to inform the user of his misconception and let him take it from there. The actual detection of the problem, however, can cause some difficulty for a parser relying heavily on semantic constraints to guide its parse. The constraint violation might cause it to assume there was some other problem such as out of order or spurious constituents, and look for (and perhaps even find) some alternative and unintended way of putting all the pieces together. This is one case where syntactic considerations should come to the fore.

Semantic constraint violations based on the mention of a related object instead of the entity actually intended by the user will manifest themselves in the same way as the semantic constraint violations based on misconceptions, but their processing needs to be quite different. The violation can be resolved if the system can look at objects related to the one the user mentioned and find one that satisfies the constraints. In the example above, this means going from the memory size to the machine that has that amount of memory. Clearly, the distance of the relationship over which this kind of substitution is allowed needs to be controlled fairly carefully – in a restricted domain everything is eventually related to everything else. But there may well be rules that control the kind of substitutions that are allowed. In the above example, it suffices to allow a part to substitute for a whole (metonymy), especially if, as we assumed, it had been used earlier in the dialogue to distinguish between different instances of the whole.

3.5. Support for recovery strategies by various parsing approaches

We now turn the question of incorporating the sentential level recovery strategies we have been discussing into the various approaches to parsing mentioned in the introduction. As we shall see, there are considerable differences in the underlying suitability of the various approaches as bases for the recovery strategies. To address this issue, we classify parsing approaches into three general groups: transition network approaches (including syntactic ATNs and network-based semantic grammars), pattern matching approaches, and approaches based on case frame instantiation.

3.5.1. Recovery strategies using a transition network approach

Although attempts have been made to incorporate sentential level recovery strategies into network-based parsers including both syntactically-based ATNs (Kwasny and Sondheimer 1981, Weischedel and Sondheimer 1984, Weischedel and Black 1980, Woods et al. 1976) and semantic grammar networks (Hendrix 1977), the network paradigm itself is not well suited to the kinds of recovery strategies discussed in the preceding sections. These strategies generally require an interpretive ability to "step back" and take a broad view of the situation when a parser's expectations are violated, and this is very hard to do when using networks. The underlying problem is that a significant amount of state information during the parse is implicitly encoded by the position in the network; in the case of ATNs, other aspects of the state are contained in the settings of scattered registers. As demonstrated by the meta-rule approach to diagnosing parse failures described by Weischedel and Sondheimer (1983) elsewhere in this journal issue, these and other difficulties elaborated below do not preclude recovery from extragrammatical input. However, they do make it difficult and often impractical, since much of the procedurally encoded state must be made declarative and explicit to the recovery strategies.

Often an ATN parse will continue beyond the point where the grammatical deviation, say an omitted word, occurred, and reach a node in the network from which it can make no further progress (that is, no arcs can be traversed). At this point, the parser cannot ascertain the source of the error by examining its internal state even if the state is accessible – the parser may have popped from embedded subnets, or followed a totally spurious sequence of arcs before realizing it was getting in trouble. If these problems can be overcome and the source of the error determined precisely, a major problem remains: in order to recover, and parse input that does not accord with the grammar, while remaining true to the network formalism, the parser must modify the network dynamically and temporarily, using the modified network to proceed through the present difficulties. Needless to say, this is at best a very complex process, one whose computational tractability is open to question. It is perhaps not surprising that in one of the most effective recovery mechanisms developed for network-based parsing, the LIFER system's ellipsis handling routine (Hendrix 1977), the key step operates completely outside the network formalism.

As we have seen, semantic constraints are very important in recovering from many types of ungrammatical input, and these are by definition unavailable

in a purely syntactic ATN parser. However, semantic information can be brought to bear on network based parsing, either through the semantic grammar approach in which joint semantic and syntactic categories are used directly in the ATN, or by allowing the tests on ATN arcs to depend on semantic criteria (Bobrow 1978, Bobrow and Webber 1980). In the former technique, the appropriate semantic information for recovery can be applied only if the correct network node can be located – a sometimes difficult task as we have seen. In the latter technique, sometimes known as cascaded ATNs (Woods 1980), the syntactic and semantic parts of the grammar are kept separate, thus giving the potential for a higher degree of interpretiveness in using the semantic information. However, the natural way to use this technique is to employ the semantic information only to confirm or disconfirm parses arrived at on syntactic grounds. So the rigidity of the network formalism makes it very difficult to bring the available semantic information to bear effectively on extragrammatical input.

A further disadvantage of the network approach for implementing flexible recovery strategies is that networks naturally operate in a top-down left-to-right mode. As we have seen, a bottom-up capability is essential for many recovery strategies, and directional flexibility often enables easier and more efficient operation of the strategies. Of course, the top-down left-to-right mode of operation is a characteristic of the network interpreter, not of the network formalism itself, and an attempt (Woods et al. 1976) has been made to operate an ATN in an "island" mode, that is, bottom-up, center-out. This experiment was done in the context of a speech parser where the low-level recognition of many of the input words was uncertain, though the input as a whole was assumed to be grammatical. In that situation, there were clear advantages to starting with islands of relative lexical certainty, and working out from there. Problems, however, arise during leftward expansion from an island when it is necessary to run the network backwards. The admissibility of ATN transitions can depend on tests that access the values of registers which would have been set earlier when traversing the network forwards, but which cannot have been set when traversing backwards. This leads at best to an increase in nondeterminism, and at worse to blocking the traversal completely.

3.5.2. Recovery strategies using a pattern matching approach

A pattern matching approach to parsing provides a better framework to recover from some sentential-level deviations than a network-based approach. In particular, the definition of what constitutes a pattern match can be relaxed to allow for missing or spurious constituents. For missing constituents, patterns which match some, but not all, of their components can be counted temporarily as complete matches, and spurious constituents can be ignored so long as they are embedded in a pattern whose other components do match. In these cases, the patterns taken as a whole provide a basis on which to perform the kind of "stepping back" discussed above as being vital for flexible recovery. In addition, when pattern elements are defined semantically instead of lexically, as with Wilks's (1975) machine translation system, semantic constraints can easily be brought to bear on the recognition. However, dealing with out of order constituents is not so easy for a pattern-based approach since constituent order is built into a pattern in a rigid way, similarly to a network. It is possible to accept any permutation of elements of a pattern as a match, but this provides so much flexibility that many spurious recognitions are likely to be obtained as well as the correct ones (see Hayes and Mouradian 1981).

An underlying problem here is that there is no natural way to make the distinctions about the relative importance or difference in role between one word and another. For instance, parsing many of the examples we have used might involve use of a pattern like:

(<determiner> <disk-drive-attribute>* <disk-drive>)

which specifies a pattern of a determiner, followed by zero or more attributes of a disk drive, followed by a phrase synonymous with "disk drive". So this pattern would recognize phrases like "a dual ported disk" or "the disk drive". Using the method of dealing with missing constituents mentioned above, "the" would constitute just as good a partial match for this pattern as "disk drive", a clearly undesirable result. The problem is that there is no way to tell the flexible matcher which components of the pattern are discriminating from the point of view of recognition and which are not. Another manifestation of the same problem is that different words and constituents may be easier or harder to recognize (for example, prepositions are easier to recognize than the noun phrases they introduce), and thus may be more or less worthwhile to look for in an attempt to recover from a grammatical deviation.

The underlying problem then is the uniformity of the grammar representation and the method of applying it to the input. Any uniformly represented grammar, whether based on patterns or networks, will have trouble representing and using the kinds of distinctions just outlined, and thus will be less well equipped to deal with many grammatical deviations in an efficient and discriminating manner. See Hayes and Carbonell (1981) for a fuller discussion of this point.

3.5.3. Recovery strategies in a case frame paradigm

Recursive case frame instantiation appears to provide

a better framework for recovery from missing words than approaches based on either network traversal or pattern matching. There are several reasons:

- Case frame instantiation is inherently a highly interpretive process. Case frames provide a high-level set of syntactic and semantic expectations that can be applied to the input in a variety of ways. They also provide an overall framework that can be used to realize the notion of "stepping back" to obtain a broad view of a parser's expectations. As we have emphasised, this ability to "step back" is important when input deviates from the standard expectations.
- Case frame instantiation is a good vehicle for bringing semantic and pragmatic information to bear in order to help determine the appropriate parse in the absence of expected syntactic constituents. If a preposition is omitted (as commonly happens when dealing with cryptic input from hunt-and-peck typists), the resulting sentence is syntactically anomalous. However, semantic case constraints can be sufficiently strong to attach each noun phrase to the correct structure. Consider, for instance, the following sentence typed to an electronic mail system natural language interface:

  "Send message John Smith"

  The missing determiner presents few problems, but the missing preposition can be more serious. Do we mean to send a message "to John Smith", "about John Smith", "with John Smith", "for John Smith", "from John Smith", "in John Smith", "of John Smith", etc.? The domain semantics of the case frame rule out the latter three possibilities and others like them as nonsensical. However, pragmatic knowledge is required to select "to John Smith" as the preferred reading (possibly subject to user confirmation) – the destination case of the verb is required for the command to be effective, whereas the other cases, if present, are optional. This knowledge of the underlying action must be brought to bear at parse time to disambiguate the cryptic command. In the XCALIBUR system case frame encoding (Carbonell, Boggs, Mauldin, and Anick 1983), we apply precisely such pragmatic knowledge represented as preference constraints (cf. Wilks 1975) on case fillers at parse time. Thus, problems created by the absence of expected case markers can be overcome by the application of domain knowledge.
- The propagation of semantic knowledge through a case frame (via attached procedures such as those of KRL (Bobrow and Winograd 1977) or SRL (Wright and Fox 1983)) can fill in parser defaults and allow the internal completion of phrases such as "dual disks" to mean "dual ported disks". This process is also responsible for noticing when information is either missing or ambiguously determined, thereby initiating a focused clarificational dialogue (Hayes 1981).
- The representation of case frames is inherently non-uniform. Case fillers, case markers, and case headers are all represented separately, and this distinction can be used by the parser interpretively instantiating the case frame. For instance, if a case frame accounts for the non-spurious part of an input containing spurious constituents, a recovery strategy can skip over the unrecognizable words by scanning for case markers as opposed to case fillers which typically are much harder to find and parse. This ability to exploit non-uniformity goes a long way to overcoming the problems with uniform parsing methods outlined in the previous section on pattern matching.

4. Dialogue Level Extragrammaticality

The underlying causes of many extragrammaticalities detected at the sentential level are rooted in dialogue phenomena. For instance, ellipses and other fragmentary inputs are patently ungrammatical at the sentential level, but can be understood in the context of a dialogue. Viewed at this more global level, ellipsis is not an "ungrammaticality". Nevertheless, the same computational mechanisms required to recover from lexical and (especially) sentential problems are necessary to detect ellipsis and parse the fragments correctly for incorporation into a larger structure. In the same way, many dialogue phenomena are classified pragmatically as extragrammaticalities.

In addition to addressing dialogue level extragrammaticalities, any robust parsing system must engage the user in dialogue for cooperative resolution of parsing problems too difficult for automatic recovery. Interaction with the user is also necessary for a cooperative parser to confirm any assumptions it makes in interpreting extragrammatical input and to resolve any ambiguities it cannot overcome on its own. We have referred several times in our discussions to the principle of focused interaction, and stated that practical recovery dialogues should be focused as tightly as possible on the specific problem at hand. Section 4.2 discusses some considerations for structuring focused interaction dialogues – in particular, why they need to be so tightly focused, and what mechanisms are needed to achieve tight focusing in a natural manner.

4.1. Ellipsis

Ellipsis is a many-faceted phenomenon. Its manifestations are varied and wide ranging, and recovery strategies for many types of ellipsis remain to be discovered. Nevertheless, it is also a very common phenomenon and must be addressed by any interface intended for serious use by real users. Empirical observations have shown that users of natural language interfaces employ

ellipsis and other abbreviating devices (for example, anaphora, short definite noun phrases, cryptic language omitting semantically superfluous words, and lexical abbreviations) with alarming frequency (Carbonell 1983). The results of our empirical observations can be summarized as follows:

> **Terseness principle:** Users of natural language interfaces insist on being as terse as possible, independent of task, communication media, typing ability, or instructions to the contrary, without sacrificing the flexibility of expression inherent in natural language communication.

Broadly speaking, one can classify ellipsis into intrasentential and intersentential ellipsis, with the latter category being far more prevalent in practical natural language interfaces. Intrasentential ellipsis occurs most frequently in coordinate clauses such as:

John likes oranges and Mary apples.

Often, this type of ellipsis is detectable only on semantic grounds (there is no meaningful noun-noun unit called "Mary apples"). The following sentence with identical syntax has a preferred reading that contains no ellipsis:

John likes oranges and MacIntosh apples.

We know of no proven general strategies for interpreting this class of intrasentential ellipsis. An interesting, but untried, approach might be an application of the strategies described below with each coordinate clause in an intrasentential ellipsis being considered as a separate utterance and with extensions to exploit the syntactic and semantic parallelism between corresponding constituents of coordinate clauses.

There are several forms of intersentential ellipsis:

- **Elaboration** – An ellipsed fragment by either speaker can be an elaboration of a previous utterance. Either speaker can make the elaboration, but the second speaker usually does so, as in the following example:

  User: Give me a large capacity disk.
  System: With dual ports?
  User: Yes, and a universal frequency adapter.

- **Echo** – A fragment of the first speaker's utterance is echoed by the second speaker. As described more fully in Hayes and Reddy (1983), this allows the second speaker to confirm his understanding of the first speaker's utterance without requiring an explicit confirmation.

  User: Add a dual disk to the order.
  System: A dual ported disk. What storage capacity?

  If, on the other hand, the system had explicitly asked "Do you mean a dual ported disk?", the user would have been conversationally obliged to reply. However, in either case, the user is free to correct any misapprehension the system displays. Sometimes, as in the example in the next bullet below, an echo may also be an expression of bewilderment. In general, this form of ellipsis is far more prevalent in spoken interactions than in typed communication, but the need for a robust parsing system to confirm assumptions it is making without being too disruptive of the flow of conversation makes it very useful for natural language interfaces in general (see Section 4.2).

- **Correction** – An ellipsed fragment substitutes for a portion of an earlier utterance that was in error. The correction occurs in three typical modes:
  - The first speaker can correct himself immediately (much like the repeated segment problem discussed in Section 3.2).
  - The second speaker can offer a correction (marked as such, or simply an ellipsed fragment in the interrogative).
  - Or, the first speaker can correct himself in response to a clarificational query from the second speaker. The form of the clarificational query can be a direct question, a statement of confusion, or echoing the troublesome fragment of the input, thereby combining two forms of ellipsis as illustrated below.

    User: Give me a dual port tape drive.
    System: A dual port tape drive?
    User: Sorry, a dual port disk drive.

- **Reformulation** – Part of an old utterance is reformulated and meant to be interpreted in place of the corresponding old constituent. This is perhaps the most common form of ellipsis and the only one for which tractable computational strategies have been implemented. All the examples below are of this type.

The LIFER/LADDER system (Hendrix 1977, Sacerdoti 1977) handled a restricted form of reformulation ellipsis. LIFER's ellipsis algorithm accepted a fragmentary input if it matched a partial parse tree derived from the previous complete parse tree by (a) selecting a subtree that accounted for a contiguous segment of the previous input, and (b) possibly pruning back one or more of its branches. If a fragmentary input matched such a partial parse tree, it was assumed to be a reformulation ellipsis and the missing parts of the partial parse tree were filled out from the previous complete parse tree. In particular, if a single grammar category accounted for the entire fragment, and this category was present in the last query parsed by the system, the ellipsis algorithm substituted the fragment

directly for whatever filled the category in the last parse. An example of this is:

User: What is the length of the Kennedy?
System: 200 meters
User: The fastest aircraft carrier?

Since both "the Kennedy" and the "the fastest aircraft carrier" match the semantic category <ship>, the latter phrase is allowed to substitute for the former. Note that a purely syntactic parse would not be sufficiently selective to make the proper substitution. "The fastest aircraft carrier" is a noun phrase, and there are three noun phrases in the original sentence: "the length", "the length of the Kennedy" and "the Kennedy".

However, the rigid structure of semantic grammars proves insufficient to handle some common forms of reformulation ellipsis. The semantic grammar formalism is too restrictive for a simple substitution strategy to apply effectively if there is more than one fragment, if there is a bridging fragment (such as "the smallest with two ports" in the example below that bridges over "disk"), or if the fragment does not preserve linear ordering. In contrast, case frame substitution provides the freedom to handle such ellipsed fragments.

The following examples are illustrative of the kind of sentence fragments the case frame method handles. We assume that each sentence fragment occurs immediately following the initial query below. Note also that we are using case frame here to refer to nominal as well as sentential case frames – the case frame being instantiated in these examples is the one for a disk with cases such as storage capacity, number of ports, etc..

INITIAL QUERY:
"What is the price of the three largest single port fixed media disks?"
SUBSEQUENT QUERIES:
"Speed?"
"Two smallest?"
"How about the price of the two smallest?"
"Also the smallest with dual ports?"
"Speed with two ports?"
"Disk with two ports?"

In these representative examples, punctuation is of no help, and pure syntax is of very limited utility. For instance, the last three phrases are syntactically similar (indeed, the last two are indistinguishable), but each requires that a different substitution be made on the preceding query.

The DYPAR-II system (discussed in Section 5.2) handles ellipsis at the case frame level. Here we present the basic case frame ellipsis resolution method it employs. Its coverage appears to be a superset of the LIFER/LADDER system (Hendrix 1977, Sacerdoti 1977) and the PLANES ellipsis module (Waltz and Goodman 1977). Although it handles most of the reformulation ellipsis we encountered, it is not meant to be a general linguistic solution to the ellipsis phenomenon.

Consider the following example:

>What is the size of the 3 largest single port fixed media disks?
>disks with two ports?

Note that it is impossible to resolve this kind of ellipsis in a general manner if the previous query is stored verbatim or as a semantic grammar parse tree. "Disks with two ports" would at best correspond to some **<disk-descriptor>** non-terminal, and hence, according to the LIFER algorithm, would replace the entire phrase "single port fixed media disks" that corresponded to **<disk-descriptor>** in the parse of the original query. However, an informal poll of potential users suggests that the preferred interpretation of the ellipsis retains the MEDIA specifier of the original query. The ellipsis resolution process, therefore, requires a finer grain substitution method than simply inserting the highest level non-terminals in the ellipsed input in place of the matching non-terminals in the parse tree of the previous utterance.

Taking advantage of the fact that a case frame analysis of a sentence or object description captures the relevant semantic relations among its constituents in a canonical manner, a partially instantiated nominal case frame can be merged with the previous case frame as follows:

- If a case is instantiated both in the original query and in the ellipsis, use the filler from the ellipsis. For instance "with two ports" overrides "single port" in our example, as both entail different values of the same case descriptor, regardless of their different syntactic roles. ("Single port" in the original query is an adjectival construction, whereas "with two ports" is a post-nominal modifier in the ellipsed fragment.)
- Retain any cases in the original parse that are not explicitly contradicted by new information in the ellipsed fragment. For instance, "fixed media" is retained as part of the disk description, as are all the sentential-level cases in the original query, such as the quantity specifier and the projection attribute of the query ("size").
- If a case is specified in the ellipsed fragment, but not in the original query, use the filler from the ellipsis. For instance, the "fixed head" descriptor is added as the media case of the disk nominal case frame in resolving the ellipsed fragment in the following example:

>Which disks are configurable on a VAX 11-780?

>Any configurable fixed head disks?

- In the event that a new case frame is mentioned in the ellipsed fragment, wholesale substitution occurs, much as in the semantic grammar approach. For instance, if after the last example one were to ask "How about tape drives?", the substitution would replace "fixed head disks" with "tape drives", rather than replacing only "disks" and producing the phrase "fixed head tape drives", which is semantically anomalous. In these instances, the semantic relations captured in a case frame representation and not in a semantic grammar parse tree prove critical.

The key advantage case frame instantiation provides for ellipsis resolution is the ability to match corresponding cases, rather than surface strings, syntactic structures, or non-canonical representations. Implementing an ellipsis resolution mechanism of equal power for a semantic grammar approach would, therefore, be very difficult. The essential problem is that semantic grammars inextricably combine syntax with semantics in a manner that requires multiple representations for the same semantic entity. For instance, the ordering of marked cases in the input does not reflect any difference in meaning,[2] while the surface ordering of unmarked cases does. With a semantic grammar, the parse trees produced by different marked case orderings can differ, so the knowledge that surface positioning of unmarked cases is meaningful, but positioning of marked ones is not, must be contained within the ellipsis resolution process. This is a very unnatural repository for such basic information. Moreover, in order to attain the functionality described above for case frames, an ellipsis resolution based on semantic grammar parse trees would also have to keep track of semantically equivalent adjectival and post nominal forms (corresponding to different non-terminals and different relative positions in the parse trees). This is necessary to allow ellipsed structures such as "a disk with 1 port" to replace the "dual-port" part of the phrase "...dual-port fixed-media disk ..." in an earlier utterance. One way to achieve this effect would be to collect together specific nonterminals that can substitute for each other in certain contexts, in essence grouping non-canonical representations into context-sensitive semantic equivalence classes. However, this process would require hand-crafting large associative tables or similar data structures, a high price to pay for each domain-specific semantic grammar. In brief, the encoding of domain semantics and canonical structure for multiple surface manifestations makes case frame instantiation a much better basis for robust ellipsis resolution than semantic grammars.

[2] leaving aside the differential emphasis and other pragmatic considerations reflected in surface ordering

4.2. Focused interaction

In addition to dealing with ellipsis and other extragrammatical phenomena that arise naturally for an interactive interface, a truly robust parsing system must initiate subdialogues of its own. Such dialogues are needed

- when a robust parser makes assumptions that may not be justified and needs confirmation from the user that it has guessed correctly;
- when a parser comes up against ambiguities that it cannot resolve on its own, either because of extragrammaticality on the part of the user or because of some essential ambiguity in perfectly grammatical input;
- when the more automated strategies may prove too costly or uncertain (e.g., when recovering from compound lexical errors);
- or when the required information is simply not present.

When an interactive system moves from the passive role of answering questions or awaiting individual user commands to a more active information-seeking role in clarificational dialogues, it must address the question of how to organize its communication so that it will behave in a way that fits with the conversational expectations and conventions of its human user. Issues of when explicit replies are required, how to convey information in such a way as to require the minimal response from the user, how to keep the conversation within the domain of discourse of the system, etc., must all be addressed by a natural language interface capable of mixed-initiative dialogue. Examining all these topics here would take us too far afield from the issue of robust parsing, so we will confine ourselves to issues specific to the kind of recovery interaction described above. See Carbonell (1982) and Hayes and Reddy (1983) for a fuller discussion of the issues involved in organizing the dialogue of an interactive natural language system.

We offer four guidelines for organizing recovery dialogues:

- the interaction should be as focused as possible;
- the required user response should be as terse as possible;
- the interaction should be in terms of the system's domain of discourse rather than the linguistic concepts it uses internally;
- there should be as few such interactions as possible.

To see the need for focused interaction, consider the input:

Add two fixed head ported disks to my order

The problem is that the user has omitted "dual" between "head" and "ported". Assuming that disks can only be single or dual ported, and using the sentential level recovery strategies described earlier, a parser

should be able to come up with an interpretation of the input that is two ways ambiguous. Interaction with the user is required to resolve this ambiguity, but the degree to which the system's initial question is focused on the problem can make a big difference in how easy it is for the user to respond, and how much work is required of the system to interpret the user response. An unfocused way of asking the question is:

Do you mean:
Add two fixed head single ported disks to my order, or
Add two fixed head dual ported disks to my order

Here the user is forced to compare two very similar looking possibilities to ascertain the system's interpretation problem. Comparisons of this kind to isolate possible interpretation problems place an unnecessary cognitive load on the user. Furthermore, it is unclear how the user should reply. Other than saying "the second one", he has little option but to repeat the whole input. Since the system's query is not focused on the source of the ambiguity, it is conversationally awkward for the user to give the single word reply, "dual". This response is highly elliptical, but from the point of view of required information, it is complete. It also satisfies our second guideline that the required response be as terse as possible.

A much better way of asking the user to resolve the ambiguity is:

Do you mean 'single' **or** 'dual' **ported disks?**

This question focuses precisely on the ambiguity, and therefore requires no effort from the user besides that of giving the information the system desires. Moreover, it invites the highly desirable reply "dual". Since the system is focused on the precise ambiguity, it can also generate a discourse expectation for this and other appropriate elliptical fragments in the user's response, and thereby recognize them with little difficulty.

The ability to generate focused queries to resolve ambiguities places certain requirements on how a parser represents the ambiguous structure internally. Unless the ambiguity is represented as locally as possible, it will be very hard to generate focused queries. If a parser finds the ambiguity in the above example by discovering it has two independent parse structures at the end of the parsing process, then generating a focused query involves a computationally taxing intractable comparison process. However, if the ambiguity is represented as locally as possible, for instance as two alternative fillers for a single instantiation of a disk frame nested within the "add to order" frame, then generating the focused query is easy – just output a paraphrase of the case frame (the one for disk) at the level immediately above the ambiguity with a disjunction taking the place of the single filler of the ambiguous case (the portedness case). Moreover, such a representation forms an excellent basis for interpreting the natural elliptical reply. As Hayes and Carbonell (1981) show, parsers based on case frame instantiation are particularly well suited to generating ambiguity representations of this kind.

Another tactic related to focused interaction that parsing systems can employ to smooth recovery dialogues is to couch their questions in terms that make it more likely that the user's reply will be something they can understand. Thus in:

Please add two 300 megabyte rotating mass storage devices to my order.

if "rotating mass storage device" is not in the system's vocabulary, it is unwise for it to reply "what is a rotating mass storage device?", since the terms the user chooses to clarify his input may be equally unintelligible to the system. It is much better to give the user a choice between the things that the system could recognize in the place where the unrecognizable phrase occurred. In this example, this would mean giving the user a choice between all the computer components that can admit 300 megabytes as a possible case filler. If this list was unmanageably long, the system should at least confirm explicitly that the unknown phrase refers to a computer component by something like:

By 'rotating mass storage device' are you referring to a computer component?

This at least establishes whether the user is trying to do something that the system can help him with or whether the user has misconceptions about the abilities of the system.

Upon confirmation that the user meant 'disk', the system could add the new phrase as a synonym for disk, perhaps after engaging the user in further clarificational dialogue to ascertain that 'disk' is not merely one kind of 'rotating mass storage device', or vice versa. If it were the case that one was more general than the other, the new entry could be placed in a semantic hierarchy and used in future recognition (perhaps after determining on what key features the two differ).

Our third guideline stated that the interaction should be in terms of the domain of discourse rather than the internal linguistic conventions of the system. Breaking this rule might involve requiring the user to, for instance, compare the parse trees representing two ambiguous interpretations of his input or telling him the name of the internal state where the parse failed in an ATN parser. Such interaction requires a linguistically and computationally sophisticated user. Moreover, it is highly non-focused from the user's point of view since it requires him to translate the parser's view of the problem into one that has meaning within the task domain, thereby switching contexts from perform-

ance of the task to linguistic issues. This enforced digression places an undue cognitive load on the user and should be avoided.

The final guideline is to minimize the amount of corrective interactions that occur. It is very tedious for a user to be confronted with questions about what he meant after almost every input, or as Codd (1974) has suggested, to approve a paraphrase of each input before the system does anything. Clearly, there are situations when the user must be asked a direct question, to wit, when information is missing or in the presence of real ambiguity. However, a technique not requiring a reply is preferable when the system makes assumptions that are very likely to be correct, or when there are strong preferences for one alternative among several in ambiguity, anaphora, or ellipsis resolution. The echoing technique mentioned in Section 4.1 is very useful in keeping required user replies to a minimum while still allowing the user to overrule any unwarranted assumptions on the part of the system. The trick is for the system to incorporate any assumptions it makes into its next output, so the user can see what it has understood, correct it if it is wrong, and ignore it if it is correct:

User: Add two dual ported rotating mass storage devices to my order
System: What storage capacity should the two dual ported disks have?

Here the system informs the user of its assumption about the meaning of "rotating mass storage device" (possible because only disks have dual ports) without asking him directly if he means "disk".

This section has given a brief glimpse of some of the dialogue issues that arise in a robust parsing system. The overriding point here is that robust parsing techniques do not stop at the single sentence level. Instead, they must be integrated with dialogue techniques that allow for active user cooperation as a recovery strategy of last resort.

5. Experiments in Robust Parsing

Having examined various kinds of extragrammaticality and the kinds of recovery strategies required to handle them, we turn finally to a series of experiments we have conducted or plan to conduct in robust parsing. Before describing some of the parsers involved in these experiments, we summarize some of the broad lessons that can be drawn from our earlier discussion. These observations have had a major role in guiding the design of our experimental systems.

- The parsing process should be as interpretive as possible. We have seen several times the need for a parsing process to "stand back" and look at a broad picture of the set of expectations (or grammar) it is applying to the input when an ungrammaticality arises. The more interpretive a parser is, the better able it is to do this. A highly interpretive parser is also better able to apply its expectations to the input in more than one way, which may be crucial if the standard way does not work in the face of an ungrammaticality.
- The parsing process should make it easy to apply semantic information. As we have seen, semantic information is often very important in resolving ungrammaticality.
- The parsing process should be able to take advantage of non-uniformity in language like that identified in Section 3.5.2. As we have seen, recovery can be much more efficient and reliable if a parser is able to make use of variations in ease of recognition or discriminating power between different constituents. This kind of "opportunism" can be built into recovery strategies.
- The parsing process should be capable of operating top-down as well as bottom-up. We have seen examples where both of these modes are essential.

Our earliest experiments in robust parsing were conducted through the FlexP parsing system (Hayes and Mouradian 1981). This system was based on partial pattern matching, and while it had the first and last of the characteristics listed above, it did not measure up well to the other two. Indeed, many of our ideas on the importance of those characteristics were developed though observation of FlexP's shortcomings as described in 3.5.2, and more fully in Hayes and Carbonell (1981). With these lessons in mind, we constructed two additional experimental parsers: CASPAR to explore the utility of case frame instantiation in robust parsing, and DYPAR to explore the notion of combining several different parsing strategies in a single parser. Both experiments proved fruitful, as the next two sections show, and DYPAR has now been developed into a complete parsing system, the DYPAR-II parser, as part of the XCALIBUR expert system interface (Carbonell et al. 1983). After that, we describe an approach to parsing we are currently developing that we believe to be based on the best features of both systems. A final section discusses other methods and approaches that we consider promising avenues for future research.

5.1. The CASPAR parser

As our earlier discussion on sentential-level ungrammaticality pointed out, case frame instantiation appears to have many advantages as a framework for robust parsing. Our initial experiments in realizing these advantages were conducted through the CASPAR parser (Hayes and Carbonell 1981). CASPAR was restricted in coverage, but could deal with simple imperative verb phrases (that is, imperative verbs followed by a sequence of noun phrases possibly marked by prepositions) in a very robust way.

Examples of grammatical input for CASPAR (drawn from an interface to a data base keeping track of course registration at a university) include:

Cancel math 247
Enrol Jim Campbell in English 324
Transfer student 5518 from Economics 101 to Business Administration 111

Such constructions are classic examples of case constructions; the verb or command is the central concept, and the noun phrases or arguments are its cases. Considered as surface cases, the command arguments are either marked by prepositions, or unmarked and identified by position, such as the position of direct object in the examples above.

The types of grammatical deviation that CASPAR could deal with include:

- Unexpected and unrecognizable (to the system) interjections as in:

  ↑S↑Q↑S[3] Enrol if you don't mind student 2476 I think in Economics 247.

- missing case markers:

  Enrol Jim Campbell Economics 247.

- out of order cases:

  In Economics 247 Jim Campbell enrol.

- ambiguous cases:

  Transfer Jim Campbell Economics 247 English 332.

Combinations of these ungrammaticalities could also be dealt with.

CASPAR used a parsing strategy specifically designed to exploit the recognition characteristics of imperative case frames, viz. that the prepositions used to mark cases are much easier to recognize than their corresponding case fillers. Below the clause level, CASPAR used linear pattern matching to recognize lower level constituents, which were defined in semantic terms appropriate to the restricted domain in which CASPAR was used. The algorithm used by CASPAR was as follows:

1. Starting from the left of the input string, apply the linear pattern matcher in scanning mode[4] using all the patterns which correspond to imperative verbs (commands). If this succeeds, the command corresponding to the pattern that matched becomes the current command, and the remainder of the input string is parsed relative to its domain-specific case frame. If it fails, CASPAR cannot parse the input.
2. If the current command has an unmarked direct object case, apply the linear pattern matcher in anchored mode at the next[5] word using the set of patterns appropriate to the type of object that should fill the case. If this succeeds, record the filler thus obtained as the filler for the case.
3. Starting from the next word, apply the pattern matcher in scanning mode using the patterns corresponding to the surface markers of all the marked cases that have not yet been filled. If this fails, terminate.
4. If the last step succeeds, CASPAR selects a marked case – the one from which the successful pattern came. Apply the matcher in anchored mode at the next word using the set of patterns appropriate to the type of object that should fill the case selected. If this succeeds record the filler thus obtained as the filler for the case.
5. Go to step 3.

Unless the input turns out to be completely unparsable, this algorithm will produce a command and a (possibly incomplete) set of arguments. It is also insensitive to spurious input immediately preceding a case marker. However, it is not able to deal with any of the other ungrammaticalities mentioned above. Dealing with them involves going back over any parts of the input that were skipped by the pattern matcher in scanning mode. If, after the above algorithm has terminated, there are any such skipped substrings, and there are also arguments to the command that have not been filled, the pattern matcher is applied in scanning mode to each of the skipped substrings using the patterns corresponding to the filler types of the unfilled arguments. This will pick up any arguments which were misplaced, or had garbled or missing case markers.

This algorithm would deal with, for instance, the convoluted example:

To Economics 247 Jim Campbell transfer please from Mathematics 121

as follows:

- The initial scan for a command verb would find "transfer", and thus cause all further parsing to be in terms of the case frame for that command.

[3] The reason for including these particular extraneous characters will be easily guessed by users of certain computers.

[4] The linear pattern matcher may be operated in *anchored* mode, where it tries to match one of a number of linear patterns starting at a fixed word in the input, or in *scanning* mode, where it tries to match the patterns it is given at successive points in the input string until one of the patterns matches, or it reaches the end of the string.

[5] The word after the last one the pattern matcher matched the last time it was applied. If some input was skipped in finding the verb in step 1, this is tacked onto the end of the sequence used by the next operation.

- The direct object required by "transfer" would not be found its expected place, after the verb, so CASPAR would skip to look for a case marker.
- The case marker "from" would be found, and CASPAR would subsequently recognize the case marked by "from" and put it in the source course slot of the transfer case frame.
- The end of the input is then reached, but some cases remain unfilled, so CASPAR goes into skipping mode looking for case markers on the missed initial segment and finds the destination course case.
- Now only the 'Jim Campbell' and 'please' segments are left and the student case is left unfilled, so CASPAR can fill the student case correctly, and has 'please' left over as spurious input.

While limited in its scope of coverage, CASPAR provides a practical demonstration of how well case frame instantiation fulfills our list of desiderata for robust parsing.

- CASPAR uses its case frames in a highly interpretive manner. It can, for instance, search directly after the verb for the filler of a case which is expected to be the direct object, but if that does not work, it is prepared to recognize the same case elsewhere in the input. Also, when it deals with out of order input, it "steps back" and takes a broad view by only considering unparsed input segments as potential fillers of cases that have not yet been filled.
- The case frame representation makes it easy to bring semantic information to bear, e.g. restrictions on what can fill each case, considerations of which cases are optional or mandatory, and whether any cases can have fillers that impose pragmatic constraints.
- CASPAR also shows the ability of case frame instantiation to exploit variations in importance and ease of recognition among different constituents. The power of exploiting such variations is shown both by the range of grammatical deviations CASPAR can handle, and by the efficiency it displays in straightforward parsing of grammatical input. This efficiency is derived from the limited number of patterns that the pattern matcher has to deal with at any one time. On its first application, the matcher only deals with command patterns; on subsequent applications, it alternates between the patterns for the markers of the unfilled cases of the current command, and the patterns for a specific object type. Also, except in post-processing of skipped input, only case marker and command patterns are employed when the pattern matcher is in its less efficient scanning mode. The constituents that are more difficult to recognize (e.g., object descriptions) are processed in the more efficient anchored mode.

Only in its predominance of top-down versus bottom-up processing does CASPAR fail to meet our desiderata. The only bottom-up component to CASPAR is the initial verb recognition phrase. If the verb were not there, it would be completely unable to parse. An extension to CASPAR to ameliorate this problem would be to start parsing case fillers bottom-up, and hypothesize the existence of the verb whose case frame most closely matched the set of case fillers found (or ask the user if there was no clear choice). This is obviously a much less efficient mode of operation than the one presented above, but it illustrates a way in which the basic case frame information could be interpreted to deal with the lack of a recognizable case header.

5.2. The DYPAR parser

DYPAR originated as an experimental vehicle to test the feasibility and potential benefits of combining multiple parsing strategies into a uniform framework. Initially, three parsing strategies (pattern matching, semantic grammar interpretation, and syntactic transformations) were combined. Transformations were used to reduce variant sentential structures to canonical form. In addition to a large set of operators,[6] the patterns could contain recursive non-terminal subconstituents corresponding to semantic grammar categories or other subconstituents. Each grammar non-terminal could expand to a full pattern containing additional non-terminals.

The experiment proved successful in that DYPAR allowed one to write grammars at least an order of magnitude more concise than pure semantic grammars of equivalent coverage. This version of the system is called DYPAR-I (Boggs, Carbonell, and Monarch 1983) and has been made available for general use. Subsequently, case frame instantiation was introduced as the new dominant strategy, and the new system, DYPAR-II, is currently used as the experimental parser for XCALIBUR, a natural language interface to expert systems (Carbonell et al. 1983).

The multi-strategy approach to parsing grammatical input in DYPAR-II facilitated the introduction of several additional strategies to recover from different kinds of extragrammaticality:

- Spelling correction combined with morphological decomposition of inflected words.
- Bridging garbled or spurious phrases in otherwise comprehensible input.
- Recognizing constituents when they occur in unexpected order in the input.
- Generalized case frame ellipsis resolution, exploiting strong domain semantics.

[6] Operators in DYPAR-I include: matching arbitrary subconstituent repetition, optional constituents, free permutation matches, register assignment and reference, forbidden constituents, and anchored and scanning modes.

The two sentential-level recovery strategies (second and third on the list above) were inspired by, and largely patterned after, the corresponding strategies in CASPAR, therefore little additional commentary is required. However, an additional complication in DYPAR-II is that the case frame instantiation process recognizes recursively embedded case frames, and in the presence of ill-formed input must deal with multiple levels of expectations. Were it not for strong domain semantics, this additional source of complexity would have introduced some ambiguity in the correction process requiring additional interaction with the user.

5.2.1. Spelling correction and morphology in DYPAR

DYPAR combines expectation-based spelling correction and morphological decomposition of inflected words. Since the DYPAR grammars are compiled into a cross-referenced form that indexes dictionary entries from patterns, it proved simple to generate lists of expected words when encountering an unrecognizable term. Although often the lists were short (highly constrained by expectations), on occasion a substantial fraction of the dictionary was generated.

Since spelling correction interacts with morphological decomposition, the two were combined into a single recovery algorithm. Here we present a somewhat simplified form of the algorithm in which the only morphological operations allowed are on word endings (e.g., singularization and other suffix stripping operations).

1. Define the *reduced dictionary* to be the set of expected words at the point in the parse where the unrecognized word was found. This set may contain expected or allowed morphological inflexions and variants, as well as root forms of words.
2. *Morphological decomposition phase* – If the word (plus any accompanying morphological information) is a member of the reduced dictionary, return it and exit.
3. Attempt to perform a one level morphological operation on the current word (e.g., stripping a legal suffix)
   a. If successful, set the word to the decomposed form (e.g. root and suffix), save the potential decomposition on a list, and go to step 2.
   b. If no morphological operation is possible, go to the spelling correction phase (step 4). Only legal sequences of suffixes are allowed.
4. *Spelling correction phase* – For each element in the list of possible decompositions (starting with the original unrecognized word), apply the spelling correction algorithm to the root word using the reduced dictionary as the candidate correction set.
   a. If successful, return the corrected word (along with any morphological information) and exit.
   b. If no spelling correction is possible, go on to the next proposed decomposition.
5. If no proposed morphological decomposition yields a recognizable root, either by direct match or spelling correction, exit the algorithm with a failure condition.

Clearly this strategy incorporates a best-match or minimal-correction criterion, rather than generating the set of all possible corrections. Moreover, words are only looked up in the reduced dictionary. This means that misspellings into words that are in the full dictionary but violate expectations (and are therefore not members of the reduced dictionary) are handled in the same manner as ordinary misspellings.

Let us trace this correction strategy on the word **"intrestingness"**. Since that word is not recognized, we enter the algorithm above and generate a reduced dictionary. Assume that the reduced dictionary contains the word **"interest"**, but none of its morphological variants. First we strip the **"ness"** suffix, but the resulting character string remains unrecognizable. Then we strip the **"ing"** suffix with similar results. Finally we strip off the coincidental **"est"** as a suffix and still find no recognizable root. At this point, morphology can do no more and the algorithm enters the spelling correction phase with the following candidate

((root: (intrestingness) suffixes: ())
 (root: (intresting) suffixes: (ness))
 (root: (intrest) suffixes: (ing ness))
 (root: (intr) suffixes: (est ing ness))

Next, we attempt to spelling correct **"intrestingness"** using the reduced dictionary and fail. We also fail with **"intresting"**, but succeed with **"intrest"** and exit the algorithm with the value

(root: (interest) suffixes: (ing ness))

and without considering the spurious **"est"** stripping. Had the word been correctly spelt, or had any of the compound morphological forms been inserted into the dictionary explicitly, the algorithm would have succeeded and exited sooner.

5.2.2. Ellipsis resolution

DYPAR-II utilizes a variant of the case frame ellipsis resolution method discussed in Section 4.1. In addition to the general algorithm, it incorporates a method for dealing with ellipsis when another component of the XCALIBUR system has generated strong discourse expectations. The ellipsed fragment is parsed in the context of these expectations, as illustrated by the recovery strategy below:

Exemplary discourse expectation rule:

IF: The system generated a query for confirmation or disconfirmation of a proposed value of a

filler of a case in a case frame in focus,
THEN: EXPECT one or more of the following:
1) A confirmation or disconfirmation pattern appropriate to the query in focus.
2) A different but semantically permissible filler of the case frame in question (optionally naming the attribute or providing the case marker).
3) A comparative or evaluative pattern appropriate to the proposed value of the case in focus.
4) A query about possible fillers or constraints on possible fillers of the case in question. [If this expectation is confirmed, a subdialogue is entered, where previously focused entities remain in focus.]

The following dialogue fragment illustrates how these expectations come into play in a focused dialogue:

>Add a line printer with graphics capabilities.

Is 150 lines per minute acceptable?

>No, 320 is better	Expectations 1, 2 & 3
(or) other options for the speed?	Expectation 4
(or) Too slow, try 300 or faster	Expectations 2 & 3

The utterance "try 300 or faster" is syntactically a complete sentence, but semantically it is just as fragmentary as the previous utterances. The strong discourse expectations suggest that it be processed in the same manner as syntactically incomplete utterances, since it satisfies the dialogue expectations listed above. Thus, the terseness principle operates at all levels: syntactic, semantic and pragmatic.

Additionally, DYPAR-II contains rules to ensure semantic completeness of utterances even in the absence of specific discourse expectations. As we have just seen, not all sentence fragments are fragmentary in the syntactic sense. But not all such purely semantic ellipsis can be predicted through dialogue generated expectations.

>Which fixed media disks are configurable on a VAX780?

The RP07**-aa, the** RP07**-ab, ...**

>Add the largest

In this example, there is no good basis for predicting what the user will do in response to the information in the answer to his question. His response turns out to be semantically elliptical – we need to answer the question "largest what?" before proceeding. One can call this problem a special case of definite noun phrase resolution, rather than semantic ellipsis, but terminology is immaterial. Such phrases occur with regularity in our corpus of examples and must be resolved by a fairly general process. The following rule answers the question from context, regardless of the syntactic completeness of the new utterance.

Contextual substitution rule

IF: A command or query case frame lacks one or more required case fillers, and the last case frame in focus has an instantiated case that meets all the semantic tests for the case missing the filler,
THEN: 1) Copy the filler onto the new case frame, and
2) Attempt to copy uninstantiated case fillers as well (if they meet semantic tests).
3) Echo the action being performed for implicit confirmation by the user.

For the example above, the case frame with a missing component is the selection case frame introduced by "largest" that requires a set of components from which to select. The previous (and therefore still focused) input has a set of disks in its only case slot and this meets the semantic criteria for the selection slot; hence it is copied over and used.

Rules such as the one above are fairly general in coverage, and the statement of the rule is independent of any specific case grammar or domain semantics. The rules, however, rely on the presence of the same specific case frames and the semantic constraints as used in the normal parsing of isolated grammatical constructions.

5.3. Multi-strategy parsing

In addition to underscoring the importance of our four desiderata for robust parsers listed at the beginning of this section, our experiments with CASPAR and DYPAR demonstrated that robustness can be achieved by the use of several different parsing strategies on the same input. These strategies operate both on grammatical input and as a means of recovery from ungrammatical input. The notion of multiple strategies fits very well with the four desiderata. In particular:

- The required high degree of interpretiveness can be obtained by having several different strategies apply the same grammatical information to the input in several different ways.
- Different strategies can be written to take advantage of different aspects of non-uniformity for different construction types.
- Some strategies can operate top-down and others bottom up.

Nor, as we have seen in DYPAR, is a multiple strategy approach inconsistent with our previous emphasis on case frame instantiation as a suitable vehicle for robust parsing. Indeed, many of the strategies required by a robust parser will be based on case frame instantiation with all the flexibility that that entails. However, case frame instantiation cannot carry the

entire burden of robustness alone, and so must be supplemented by other strategies such as the ones present in DYPAR. In fact, even the method of case frame instantiation presented for CASPAR can be seen as two strategies: one an initial pass using standard expectations, and the other a recovery strategy for when the first fails. The bottom-up strategy discussed at the end of the section on CASPAR would make a third.

5.3.1. Coordinating multiple strategies through an entity-oriented approach

A major problem that arises in using multiple parsing strategies is coordination between the strategies. Questions of interaction and order of application are involved. In CASPAR and DYPAR, the problem was solved simply by "hard-wiring" the interactions, but this is not satisfactory in general, especially if we wish to extend the set of strategies available in a smooth way. One alternative we have begun to explore involves the idea of entity-oriented parsing (Hayes 1984).

The central notion behind entity-oriented parsing is that the primary task of a natural language interface is to recognize entities – objects, actions, states, commands, etc. – from the domain of discourse of the interface. This recognition may be recursive in the sense that descriptions of entities may contain descriptions of subsidiary entities (for example, commands refer to objects).

In entity-oriented parsing, all the entities that a particular interface system needs to recognize are defined separately. These definitions contain information both about the way the entities will be manifested in the natural language input (this information can also be used to generate output), and about the internal semantic structure of the entities. This arrangement has the following advantages for parsing robustness:

- The individual entity definitions form an ideal framework around which to organize multiple parsing strategies. In particular, each definition can specify which strategies are applicable to recognizing it. Of course, this only provides a framework for robust recognition, the robustness achieved will still depend on the quality of the various recognition strategies used.
- The individual definition of all recognizable domain entities allows them to be recognized independently. Assuming there is appropriate indexing of entities through lexical items that might appear in a surface description of them, this recognition can be done bottom-up, thus allowing for recognition of elliptical, fragmentary, or partially incomprehensible input. The same definitions can also be used in a more efficient top-down manner when the input conforms to the system's expectations.
- This style of organization is particularly well suited to case frame instantiation. The appropriate case frames can be associated with each entity definition for use by case-oriented strategies. Of course, this does not prevent other strategies from being used to recognize the entity, so long as suitable information for the other strategies to interpret is provided in the entity definition.

These arguments can be made more concrete by example.

5.3.2. Example entity definitions

First we examine some example entity and language definitions suitable for use in entity-oriented parsing. The examples are drawn from the domain of an interface to a data base of college courses. Here is the (partial) definition of a course. Square brackets denote attribute/value lists, and round brackets ordinary lists.

```
[
EntityName: CollegeCourse
Type: Structured
Components: (
   [ComponentName: CourseNumber
     Type: Integer
     GreaterThan: 99
     LessThan: 1000
   ]
   [ComponentName: CourseDepartment
     Type: CollegeDepartment
   ]
   [ComponentName: CourseClass
     Type: CollegeClass
   ]
   [ComponentName: CourseInstructor
     Type: CollegeProfessor
   ]
   ...
    )
SurfaceRepresentation: (
   [SyntaxType: Pattern
    Pattern: ($CourseDepartment $CourseNumber)
   ]
   [SyntaxType: NounPhrase
    Head: (course | seminar | ...)
    AdjectivalComponents: (CourseDepartment ...)
    Adjectives: (
      [AdjectivalPhrase: (new | most recent)
       Component: CourseSemester
       Value: CurrentSemester
      ]
      ...
       )
   PostNominalCases: (
      [Case-marker: (?intended for | directed to | ...)
       Component: CourseClass
```

```
      ]
      [Case-marker: (?taught by | ...)
        Component: CourseInstructor
      ]
      ...
        )
  ]
  ...
    )
]
```

Precise details of this language are not relevant here. Important features to note include the definition of a course as a structured object with components: number, department, instructor, etc.. This definition is separate from the surface representation of a course which is defined to take one of two forms: a simple word pattern of the course department followed by the course number (dollar signs refer back to the components), or a full noun phrase with adjectives, postnominal cases, etc. Since we are assuming a multi-strategy approach to parsing, the two quite different kinds of surface language definition do not cause any problem – they can both be applied to the input independently by different construction-specific strategies, and the one which accounts for the input best will be used.

Subsidiary objects like CollegeDepartment are defined in similar fashion.

```
[
EntityName: CollegeDepartment
Type: Enumeration
EnumeratedValues: (
   ComputerScienceDepartment
   MathematicsDepartment
   HistoryDepartment
   ...
    )
SurfaceRepresentation: (
   [SyntaxType: Pattern
     Pattern: (CS | Computer Science | Comp Sci | ...)
     Value: ComputerScienceDepartment
   ]
   ...
    )
]
```

CollegeCourse itself will be a subsidiary entity in other higher-level entities of our restricted domain, such as a command to the data base system to enrol a student in a course.

```
[
EntityName: EnrolCommand
Type: Structured
Components: (
   [ComponentName: Enrollee
     Type: CollegeStudent
   ]
   [ComponentName: EnrollIn
     Type: CollegeCourse
   ]
    )
SurfaceRepresentation: (
   [SyntaxType: ImperativeCaseFrame
     Head: (enrol | register | include | ...)
     DirectObject: ($Enrollee)
     Cases: (
       [Case-marker: (in | into | ...)
         Component: EnrollIn
       ]
        )
   ]
    )
]
```

5.3.3. Parsing with an entity-oriented approach

Now we turn to the question of how language definitions like those in the examples just given can be used to drive a parser. Let us examine first how a simple data base command like

Enrol Susan Smith in CS 101

might be parsed using the above language definitions. The first job is to recognize that we are parsing an EnrolCommand. In a purely top-down system, we would establish this by having a list of all the entities that we are prepared to recognize as complete inputs and trying each one of these to see if they could be recognized, a rather inefficient process. A more natural strategy in an entity-oriented approach is to try to index bottom-up from words in the input to those entities that they might appear in. In this case, the best indexer for EnrolCommand is the first word, 'enrol'. In general, the best indexer need not be the first word of the input and we need to consider all words, thus raising the potential of indexing more than one entity. Hence we might also index CollegeStudent, CollegeCourse, and CollegeDepartment. A simple method of cutting down the number of index-generated possibilities to investigate top-down is to eliminate all those that are subsidiary to others that have been indexed. For our example, this would eliminate everything except EnrolCommand, the desired result. One final point about indexing: it is clearly undesirable to index from every word that could appear in the surface representation of an entity; only highly discriminating words like 'enrol' or 'CS' should be used. Whether a word is sufficiently discriminating can be determined either manually, which is unreliable, or automatically by keeping a count of the number of entities indexed by a given word and removing it from the index if it indexes more than a certain threshold number.

Once EnrolCommand has been established as the entity to recognize in the above example, the remainder of the recognition can be accomplished straightforwardly in a top-down manner. The definition of the surface representation of EnrolCommand is an imperative case frame with a CollegeStudent as direct object and with a CollegeCourse as a second case indicated by 'in'. This information can be used directly by a simple case frame recognition strategy of the type used in CASPAR. No translation into a structurally different representation is necessary. The most natural way to represent the resulting parse would be:

```
[InstanceOf: EnrolCommand
  Enrollee: [InstanceOf: CollegeStudent
             FirstNames: (Susan)
             Surname: Smith
             ]
EnrolIn:  [InstanceOf: CollegeCourse
           CourseDepartment: ComputerScienceDepartment
           CourseNumber: 101
          ]
]
```

Note how this parse result is expressed in terms of the underlying structural representation used in the entity definitions without the need for a separate semantic interpretation step.

To see the possibilities for robustness with the entity-oriented approach, consider the input:

> Place Susan Smith in computer science for freshmen

There are two problems here: we assume that the user intended 'place' as a synonym for 'enrol', but that it happens not to be in the system's vocabulary; the user has also shortened the grammatically acceptable phrase, 'the computer science course for freshmen', to an equivalent phrase not covered by the surface representation for CollegeCourse as defined above. Since 'place' is not a synonym for 'enrol' in the language as presently defined, we cannot index EnrolCommand from it and hence cannot get the same kind of top-down recognition as before. So we are forced to recognize smaller fragments bottom-up. Let's assume we have a complete listing of students and so can recognize 'Susan Smith' as a student. That leaves 'computer science for freshmen'. We can recognize 'computer science' as a CollegeDepartment and 'freshmen' as a CollegeClass, so since they are both components of CollegeCourse, we can attempt to unify our currently fragmentary recognition by trying to recognize a course description from the segment of the input that they span, viz. 'computer science for freshmen'.

There are two possible surface representations given for CollegeCourse. The first, a pattern, is partially matched by 'computer science', but does not unify the two fragments. The second, a noun phrase accounts for both of the fragments (one is adjectival, the other part of a post-nominal case), but would not normally match them because the head noun is missing. In fragment recognition mode, however, this kind of gap is acceptable, and the phrase can be accepted as a description of a CollegeCourse with ComputerScienceDepartment as CourseDepartment, and FreshmanClass as CourseClass.

The input still consists of two fragments, however, a CollegeStudent and a CollegeCourse, and since we do not have any information about the word 'place', we are forced to consider all the entities that have those two sub-entities as components. We will suppose there are three: EnrolCommand, WithdrawCommand, and TransferCommand (with the obvious interpretations). Trying to recognize each of these, we can rule out TransferCommand in favour of the first two because it requires two courses and we only have one. Also, EnrolCommand is preferred to WithdrawCommand since the preposition 'in' indicates the EnrolIn case of EnrolCommand, but does not indicate WithdrawFrom, the course-containing case of WithdrawCommand. Thus we can conclude that the user intended an EnrolCommand.

In following this bottom-up fragment combination procedure, we have ignored other combination possibilities that did not lead to the correct answer – for instance, taking 'Computer Science' as the StudentDepartment case of the CollegeStudent, 'Susan Smith'. In practice, an algorithm for bottom-up fragment combination would have to consider all such possibilities. However, if, as in this case, the combination did not turn out to fit into a higher-level combination that accounted for all of the input, it could be discarded in favour of combinations that did lead to a complete parse. More than one complete parse would be handled, just like any other ambiguity, through focused interaction.

Even assuming that the above example had a unique result, since it involved several significant assumptions, we would need to use focused interaction techniques (Hayes 1981) to present a paraphrase of our interpretation to the user for approval before acting on it. Note that if the user does approve it, we should be able (perhaps with further approval) to add 'place' to the vocabulary as a synonym for 'enrol' since 'place' was an unrecognized word in the surface position where 'enrol' should have been.

A pilot implementation of a parser constructed according to the entity-oriented principles outlined above has been completed and preliminary evaluation is promising. We are hoping to build a more complete parser along these lines.

6. Concluding Remarks

Any practical natural language interface must be capable of dealing with a wide range of extragrammatical

input. This paper has proposed a taxonomy of the prevalent forms of extragrammaticality in real language use and presented recovery strategies for many of them. We also discussed how well various approaches to parsing could support the recovery strategies, and concluded that case frame instantiation provided the best framework among the commonly used parsing methodologies.

At a more general level, we argued that the superiority of case frame instantiation over other parsing methodologies for robust parsing is due to how well it satisfies four parsing characteristics that are important for many of the recovery strategies that we described:

- The parsing process should be as interpretive as possible.
- The parsing process should make it easy to apply semantic information.
- The parsing process should be able to take advantage of non-uniformity in language.
- The parsing process should be capable of operating top-down as well as bottom-up.

We claimed that while case frame instantiation satisfies these desiderata better than any other commonly used parsing methodology, it was possible to do even better by using a multi-strategy approach in which case frame instantiation was just one member (albeit a very important one) of a whole array of parsing and recovery strategies. We described some experiments that led us to this view and outlined a parsing methodology, entity-oriented parsing, that we believe will support a multi-strategy approach.

It is our hope that by pursuing lines of research leading to parsers that maximize the characteristics listed above, we can approach, in semantically limited domains, the extraordinary degree of robustness in language recognition exhibited by human beings, and gain some insights into how robustness might be achieved in more general language settings.

7. References

Bobrow, D.G. and Winograd, T. 1977 An Overview of KRL, a Knowledge Representation Language. *Cognitive Science* 1(1): 3-46.

Bobrow, R.J. 1978 The RUS System. BBN Report 3878. Bolt Beranek and Newman, Cambridge, Massachuseets.

Bobrow, R.J. and Webber, B. 1980 Knowledge Representation for Syntactic/Semantic Processing. *Proc. National Conference of the American Association for Artificial Intelligence.* Stanford University, Stanford, California (August).

Bobrow, D.G.; Kaplan, R.M.; Kay, M.; Norman D.A.; Thompson, H.; and Winograd, T. 1977 GUS: a Frame-Driven Dialogue System. *Artificial Intelligence* 8: 155-173.

Boggs, W.M. and Carbonell, J.G. and Monarch, I. 1983 The DYPAR-I Tutorial and Reference Manual. Technical report. Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

Brown, J.S. and Burton, R.R. 1975 Multiple Representations of Knowledge for Tutorial Reasoning. In Bobrow, D.G. and Collins, A., eds. *Representation and Understanding.* Academic Press, New York, New York: 311-349.

Carbonell, J.G. 1979 Towards a Self-Extending Parser. *Proceedings of the 17th Meeting of the Association for Computational Linguistics:* 3-7.

Carbonell, J.G. and Hayes, P.J. 1984 Robust Parsing Using Multiple Construction-Specific Strategies. In Bolc, L., ed., *Natural Language Parsing Systems.* Springer-Verlag, New York, New York.

Carbonell, J.G.; Boggs, W.M.; Mauldin, M.L.; and Anick, P.G. 1983 The XCALIBUR Project, A Natural Language Interface to Expert Systems. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence.*

Carbonell, J.G.; Boggs, W.M.; Mauldin, M.L.; and Anick, P.G. 1983 XCALIBUR Progress Report #1: First Steps Towards an Integrated Natural Language Interface. Technical report. Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

Carbonell, J.G. 1982 Meta-Language Utterances in Purposive Discourse. Technical report. Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

Carbonell, J.G. 1983 Discourse Pragmatics in Task-Oriented Natural Language Interfaces. *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics.*

Codd, E.F. 1974 Seven Steps to RENDEZVOUS with the Casual User In Klimbie, J.W. and Koffeman, K.L., eds., *Proceedings of the IFIP TC-2 Working Conference on Data Base Management Systems.* North Holland, Amsterdam: 179-200.

Dejong, G. 1979 Skimming Stories in Real-Time. Ph.D. dissertation. Computer Science Department, Yale University, New Haven, Connecticut.

Durham, I.; Lamb, D.D.; and Saxe, J.B. 1983 Spelling Correction in User Interfaces. *Communications of the ACM* 26.

Haas, N. and Hendrix, G.G. 1983 Learning by Being Told: Acquiring Knowledge for Information Management. In Michalski, R.S.; Carbonell, J.G.; and Mitchell, T.M., eds., *Machine Learning, An Artificial Intelligence Approach.* Tioga Press, Palo Alto, California.

Hayes, P.J. 1981 A Construction Specific Approach to Focused Interaction in Flexible Parsing. *Proceedings of 19th Annual Meeting of the Association for Computational Linguistics* (June): 149-152.

Hayes, P.J. 1984 Entity-Oriented Parsing. COLING84, Stanford University, Stanford, California (July).

Hayes, P.J. and Mouradian, G.V. 1981 Flexible Parsing. *American Journal of Computational Linguistics* 7(4); 232-241.

Hayes, P.J. and Carbonell, J.G. 1981 Multi-strategy Construction-Specific Parsing for Flexible Data Base Query and Update. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence.* Vancouver (August): 432-439.

Hayes, P.J. and Reddy, D.R. 1983 Steps Toward Graceful Interaction. *Spoken and Written Man-Machine Communication, International Journal of Man-Machine Studies.* 19(3): 211-284.

Hayes, P.J. and Carbonell, J.G. 1981 Multi-Strategy Parsing and its Role in Robust Man-Machine Communication. Technical report CMU-CS-81-118. Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania (May).

Hendrix, G.G. 1977 Human Engineering for Applied Natural Language Processing. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence:* 183-191.

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.

Sacerdoti, E.D. 1977 Language Access to Distributed Data with Error Recovery. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence:* 196-202.

Schank, R.C.; Lebowitz, M.; and Birnbaum, L. 1980 An Integrated Understander. *American Journal of Computational Linguistics* 6(1): 13-30.

Waltz, D.L. 1978 An English Language Question Answering System for a Large Relational Data Base, *Communications of the ACM* 21(7): 526-539.

Waltz, D.L. and Goodman, A.B. 1977 Writing a Natural Language Data Base System. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence:* 144-150.

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

Weischedel, R.M. and Black, J. 1980 Responding to Potentially Unparseable Sentences. *American Journal of Computational Linguistics* 6: 97-109.

Wilks, Y. A. 1975 Preference Semantics. In Keenan, ed., *Formal Semantics of Natural Lanbuage.* Cambridge University Press, Cambridge, England.

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

Woods, W.A.; Kaplan, R.M.; and Nash-Webber, B. 1972 The Lunar Sciences Language System: Final Report. Technical report 2378. Bolt Beranek and Newman, Cambridge, Massachusetts.

Woods, W.A.; Bates, M.; Brown, G.; Bruce, B.; Cook, C.; Klovstad, J.; Makhoul, J.; Nash-Webber, B.; Schwartz, R.; Wolf, J.; and Zue, V. 1976 Speech Understanding Systems – Final Technical Report. Technical report 3438. Bolt Beranek and Newman, Cambridge, Massachusetts.

Wright, K. and Fox, M 1983 The SRL Users Manual. Technical report. Robotics Institute, Carnegie-Mellon University, Pittsburgh, Pennsylvania.