Multi-Strategy Construction-Specific Parsing for Flexible Data Base Query and Update

Philip J. Hayes and Jaime G. Carbonell

Carnegie-Mellon University
Pittsburgh, PA 15213, U. S. A.

Abstract

The advantages of a multi-strategy, construction-specific approach to parsing in applied natural language processing are explained through an examination of two pilot parsers we have constructed. Our approach exploits domain semantics and prior knowledge of expected constructions, using multiple parsing strategies each optimized to recognize different construction types. It is shown that a multi strategy approach leads to robust, flexible, and efficient parsing of both grammatical and ungrammatical input in limited-domain, task oriented, natural language interfaces. We also describe plans to construct a single, practical, multi-strategy parsing system that combines the best aspects of the two simpler parsers already implemented into a more complex, embedded-constituent control structure. Finally, we discuss some issues in data base access and update, and show that a construction-specific approach, coupled with a case-structured data base description, offers a promising approach to a unified, interactive data base query and update system.[1]

1. Introduction

Providing robust natural language interfaces to interactive computer systems is a rapidly growing concern in natural language processing. Much of the work in this area has focused on parsing problems that arise in applied natural language processing, and in particular, on mechanisms to exploit strong domain-dependent semantic constraints. Past work in this area includes LIFER [12], SOPHIE [3, 4], LUNAR [24], and PLANES [21]. Other investigators have concentrated on handling the performance errors that inevitably occur in spontaneously-used language (Hayes and Mouradian [10], Weischedel and Black [22], and Kwasny and Sondheimer [16]). Ungrammatical input was also a major concern of Colby in the PARRY system [18], and Wilks [23] in parsing input with non standard semantic relations. All of these efforts, .however, have followed the paradigm of applying a uniform parsing procedure to a uniformly represented grammar, failing to exploit domain specific constructions and not always using the powerful domain semantics to best effect. Although the parsing procedures were flexible enough to deal with certain forms of ungrammatical input, they were limited by having to use the same uniform techniques on all types of construction, and hence could not take advantage of specific features of individual construction types. Moreover, a uniform-grammar approach requires that domain semantics be coerced and simplified to fit a predetermined mold, thus limiting the scope and utility of task-specific knowledge in the parsing process.

Our objective in this work is to produce *task oriented, natural language interfaces that are robust, flexible, and efficient.* Therefore, we plan to develop, refine, and test a number of different mechanisms, each designed to perform its own particular task reliably and efficiently. We intend to exploit all possible tools at our disposal in creating a system capable of selecting the tool best suited for the job at hand. Thus, we advocate a "tool-chest" of parsing and representational techniques for the parser to apply, rather than trying to design a single multi purpose tool. This objective differs from both cognitive modelling approaches (e.g. [2]) and elegant linguistic solutions (e.g. [17]) where integration (in the former) and uniformity (in the latter) are primary considerations.

We have argued elsewhere [6,8] at some length in favor of parsing strategies, grammar representations, and domain semantics that are *construction-specific* rather than uniform. In other words, we argued that for each type of construction in a language, there should be a specific formalism for representing constructions of that type, plus a specific procedure for applying instances of each type of construction to the parsers input. The parser would switch among the various parsing strategies dynamically depending on the input. Examples of what we mean by construction type include case constructions (e.g. in imperative commands and the macro structure of noun phrases with post-nominal modifiers), conjoined constructions. positional constructions such as simple noun phrases, and non standard constructions such as names, times, and addresses.

Rather than reiterating our reasons for advocating a construction specific approach, we list below the main benefits claimed for such an approach, and refer the reader to [6, 8] for full justifications.

- Different constituents of a given construction can serve quite different functions and exhibit radically different ease of recognition. In a case construction, for instance, the case markers carry information about relations, whereas the case fillers describe the objects being related; also the case markers are typically drawn from a limited set of possibilities and are consequently much easier to recognize than the case fillers, which typically exhibit much more variety. Construction specific parsing techniques are able to capitalize on these distinctions when ungrammatical input is present. For example, a failed parse of a case construction may be restarted by scanning the input for one of the easily recognized case markers, thus temporarily skipping the incomprehensible segment.

This research was sponsored jointly by the Defense .Advanced Research Projects Agency (DOD). ARPA Order No 3597. monitored by the Air Force Avionics Laboratory Under Contract F33615 78-C 1551. and by the Air Force Office of Scientific Research under Contract. F49620 79 C0143 The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency, the Air Force Office of Scientific Research, or the US Government

- Because complicated constructions, such as case constructions, can be represented and applied as a whole to the input, rather than being split up over a number of rules or network segments as in a uniform-'grammar approach to parsing, problems that arise with ungrammatical instances of case constructions can be dealt with by a strategy that can take into account any fragments of the construction that have already been parsed. Thus, if certain cases have been recognized and instantiated, the troublesome segment in the input need only be matched with the uninstantiated cases. This enables the parser to apply much stronger semantic constraints and structural expectations, thus easing the burden on the syntax and case-selection processes [5,6].
- A construction-specific approach allows highly efficient parsing of grammatical input by providing an excellent framework for the application of the strong typing constraints characteristic of limited domains.
- Regardless of the structure being parsed into, structural ambiguities will sometimes arise from the input. A construction specific approach provides two powerful mechanisms to handle ambiguity: 1) Strong semantic constraints from the application domain rule out most common ambiguities. 2) When ambiguities arise (often due to ellipsed or ungrammatical utterances), the construction specific approach facilitates an explicit and localized representation of ambiguity, without duplication of unambiguous parts of the input. This is a tremendous aid to an interface in presenting the user with a request for clarification. The request can be focused on the precise nature of the problem that the system experienced with the input, and the system is in a better position to understand elliptical responses from the user [8].
- A construction-specific approach also helps in the definition and development of languages for limited domain interfaces. Because the constructions dealt with are those that are "natural[0] for the task domain, a language definition expressed in terms tied closely to the structure of the task domain can be interpreted directly by construction-specific strategies without the need for an intermediate, time consuming, compilation phase into a uniform-grammar formalism. This greatly speeds the testing of the many small changes that inevitably have to be made in the course of developing a language, and so makes the language designer's job significantly easier.

In this paper, we examine two small parsers we have constructed in order to illustrate some of these benefits, and to serve as stepping stones to the construction of a larger parsing system, which will integrate all the features we have mentioned. The first of these simple parsers shows the power of a construction specific approach in processing ungrammatical input; it is oriented around case constructions, and uses the distinctive characteristics of such constructions to deal with grammatical deviations in an extremely robust fashion. The second parser is a good illustration of the advantages of switching between parsing strategies dynamically; by combining only three very simple parsing strategies, it is able to deal with a surprisingly wide range of input. We go on to discuss combining the advantages of these two, essentially "toy", parsers into a single, useful multi-strategy parsing system. We also discuss further advantages of a construction-specific approach when it is used in an interface for accessing and updating data bases.

2. The CASPAR Parser

In this section we examine CASPAR, a small parser we constructed as an illustration of the power of a construction-specific approach in dealing with ungrammatical input. It also turned out to provide a very efficient way of recognizing grammatical input in the class of domain specific languages for which it was designed.

CASPAR was designed to provide a natural language command interface to an interactive computer system. Since an imperative is a natural way to issue system commands, CASPAR was designed to recognize simple imperative verb phrases, i.e. imperative verbs followed by a sequence of noun phrases possibly marked by prepositions. Examples for an interface to a data base keeping track of registration for college courses include:

cancel math 247
enroll Jim Campbell in English 324
transfer student 5518 from Physics 101 to comp sci 111

The imperative verbs identify the system commands and the noun phrases provide their arguments. 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 preposition, or unmarked and identified by position such as the position of direct object in the examples above.

In line with in the construction-specific approach we .are advocating, CASPAR was given two quite distinct parsing strategies:

- A strategy to identify the appropriate case frame and activate its case markers and filler-patterns to deal with the rest of the input utterance.
- A strategy to recognize individual constituent case filters and markers, including the verb, noun phrases in the role of case fillers, and prepositions in the role of case markers.

The first of these strategies is dominant in the sense that it decides where in the input the second, more detailed, recognizer should be applied and what it should try to recognize when it is applied. The second strategy is a simple linear pattern matcher. This is just what is needed for verbs, prepositions, and simple object descriptions such as those in the examples above, but it is inadequate for more complicated kinds of object descriptions, and in particular, for object descriptions that are themselves case constructions as in:

cancel the classes taught by Solway on Tuesday

This deficiency is what relegates CASPAR to the realm of toy systems. However, see [6] for the design of a multi strategy parser that can deal with nested case constructions.

While CASPAR is just an experimental system, the flexibility and robustness obtained by providing separate parsing strategies for the two different construction types it recognizes (case and fixed-order linear patterns) is quite striking. The types of grammatical deviation that can be dealt with include:

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

  tStQtS[2]enroll student 2476 in I think CS 348.

- missing case markers:

  enroll Jim Campbell Economics 247.

- out of order cases:

  In Economics 247 Jim Campbell enroll.

The reason for including these particular extraneous characters wilt be easily guessed by users of certain computer systems

• ambiguous cases:

transfer Jim Campbell Economics 247 English 332.

Combinations of these ungrammaticalities can also be dealt with.

CASPAR achieves this degree of robustness by exploiting certain specialized characteristics of case constructions; most importantly, it takes advantage of the differences between case markers and case fillers. Case markers are typically drawn from a small set of words or phrases, and are thus much easier to recognize (or spelling correct) than case fillers, which have much more variety. This ease of recognition of case markers makes it practical for CASPAR to scan the entire input for them, and thus to locate and parse the corresponding case fillers; interjections and out of order cases are dealt with in this way. CASPAR also keeps track of which cases have been filled, and thus cuts down on the number of filler types that it has to try when a segment of input must be parsed without the guidance of case markers, for instance, if some case markers have been omitted. Neither of these heuristics are available to the more uniform parsing procedures of Weischedel and Black (22). or Kwasny and Sondheimer [16)), or of our own FlexP parser [10], simply because there is no convenient way for these parsers to represent or make use of the information that case markers are much easier to recognize than case fillers or that most cases can be filled only once. Exactly how these heuristics operate, along with other details of how CASPAR is tailored to case constructions, can be seen from the following description of CASPARS parsing algorithm.

CASPAR has two parsing strategies: a case-oriented strategy and a linear pattern matching strategy. The case oriented strategy controls the operation of the pattern matching strategy, which in turn actually recognizes words from the input. 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. The case-oriented parsing strategy operates in the following way.

1. Starting from the left of the input string, apply the linear pattern matcher in scanning mode using all the patterns which correspond to 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 matcher in anchored mode at the next[3] 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.

[3]The word after the last one the pattern matcher matched the leat time it was applied.

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 have been skipped by using 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. If one of the arguments matched in this way could fill more than one slot, a special representation is used for CASPAR'S output indicating the ambiguity without duplicating unambiguous parts of the parse This representation, which has advantages in formulating requests to the user to resolve the ambiguity, is described in more detail in [8].

The grammar description that CASPAR uses is tied very closely to the structure of the domain. For each possible command to the underlying system, the grammar definition contains a list of the linear patterns which can be used to refer to that command, plus a list of arguments to the command. For each argument the definition gives the type of domain object that should fill that argument, plus the linear patterns used as surface case markers to signal the argument (or an indication that the argument is an unmarked case such as the direct object). The grammar definition also gives the linear patterns needed to recognize each type of domain object. See Section 5 below for an example of a grammar definition in a similar style, and for a discussion of how a construction-specific approach to parsing fits well with a grammar definition that is tied closely to domain structure. This point is also discussed in [8]. This form of grammar definition fits naturally into a description of the underlying interactive system as a whole. Such a description can be used to control other aspects of a cooperative and graceful user interface • see [1] for more details of other work we have done in that area.

While simplistic in many ways, CASPAR shows the power of a construction-specific approach to parsing, both in the range of grammatical deviations it can handle, and in 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 more difficult to recognize object descriptions are processed in the more efficient anchored mode. This efficiency is sound evidence that such a construction-specific approach is a good way to bring the powerful semantic restrictions available *in* limited domains to bear on the parsing of both grammatical and ungrammatical input.

3. The DYPAR Parser

As a related investigation of the practical feasibility of dynamic strategy selection by a domain oriented parser, we also developed the DYPAR[4] system. DYPAR has a *Kernel control module* to select the appropriate parsing strategy as a function of the expected input structure, plus three parsing strategies to select among, each with its own grammatical and/or semantic knowledge encodings, and global data structures to share information. The control structure, strategies, and linguistic knowledge representations are augmented with domain specific semantic knowledge bases. Thus, the same kernel parser may be applied to different domains if a detailed, domain specific, semantic knowledge base is provided.

Whereas the central focus of CASPAR was to exploit domain semantics and construction specific case frames for processing some types of malformed input, DYPAR was built to explore issues of user interaction and multi strategy synthesis in the context of a working parser. More explicitly, our objectives for developing DYPAR were threefold:

- Test the feasibility of a multi-strategy approach.
- Investigate a simple data base task requiring interaction and feedback between the system and the user.
- Eventually integrate the best features of CASPAR and DYPAR into a robust, construction specific, multi strategy parser.

In encoding domain semantics, we found that some information can be expressed more naturally and parsimoniously in one form (e.g., linear patterns), while other information is best expressed in other forms (eg, equivalence transformations or semantic grammar productions). To illustrate this point, we attempted to encode all the knowledge in DYPAR as a pure semantic grammar. This task has more than tripled the size of the task-specific knowledge base, and we have not yet finished (nor do we intend to finish) the conversion. The primary reason for the increase in size is that much of the information must be stated with a high degree of redundancy and often in an awkward, roundabout manner when it is coerced into a uniform, context-free representation. Therefore, the primary lesson one can draw from the DYPAR effort is that multi-strategy parsing is tractable in practice and moreover can perform the work of single strategy approaches with much greater economy of programmer effort.[5]

3.1. Parsing Strategies in DYPAR

DYPAR combines three parsing strategies:

- A context-free semantic grammar component, grouping domain information into hierarchical semantic categories useful in classifying individual words and phrases in the input language, similar to the LIFER semantic grammar mechanism [12].
- A partial pattern match component, represented as pattern action rules. The patterns may contain individual words, semantic categories (from the semantic grammar), wild cards, optional constituents, register assignment and register reference. This method enables the semantic grammar non-terminal categories to be applied in a much more effective context-sensitive manner than in a pure context-free grammar recognizer.
- Equivalence transformations map domain dependent and domain-independent constructs into canonical form, requiring a fraction of the patterns and semantic categories that would otherwise be needed. If a phrase-structure can be expressed in several different ways, while retaining the same meaning, it is clearly beneficial to first map it into canonical form, rather than being forced to include all possible variants in every context where that constituent could occur.

Below we give an example of each type of linguistic information used in DYPAR. In order to understand these examples, a few notational conventions must be introduced: <BRACKETS> denote a non-terminal semantic grammar symbol. A word starting with an exclamation mark (e.g.. {REGISTER) denotes the name of register. A vertical bar (|) denotes disjunction in a pattern. A # in a pattern matches a single word. An asterisk (•) matches an arbitrary sequence of words. The construction (JREGISTER pattern) assigns whatever matches the pattern to the register specified. A question mark (?) before a constituent in a pattern indicates that constituent is optional.

DYPAR, as we see in the dialogue below, is the front end of a semantic network data base update and query system. Therefore, its domain knowledge consists of language constructs relevant to this task.'First, consider a fragment of its semantic grammar:

```
<INFO-REQ>  ->  (<WHAT-Q>  |  <INF0-REQ1>)
<INF0-REQ1>  ->  (?<POLITE>  <INF0-REQ2>
                  7<WHAT-Q>)
<INF0-REQ2>  ->  (TELL  <me-US>  ?ABOUT  |
                  GIVE  <me-US>  |  PRINT  |  TYPE)
```

This fragment, together with the rewrite rules for the other non-terminals above (e.g., <BEPRES>, whose rewrite is alt the present-tense conjugations of the verb "to be") recognizes the initial segment of information request queries such as: "What is ...", "Tell me what is ...", "Tell me about...", "Would you give me ..", etc.

Now, consider a pattern-match rule:

```
(?<det>  (Ival #)  <be-pre$>  ?<DET>  (IPROP •#)
          OF  ?<DET>  (INAM #)  ?<dpunct>)
■>
(LTM-STORE  INAM  IVAL  IPROP)
```

This rule recognizes sentences such as: "Felix is a friend of Fido", or "Reagan is president of the USA", and passes the information to the data base manager for consistency checking and storage. In order to pass the information gathered in the pattern match process, the registers are assigned appropriate values. For instance, in the second example, INAM is assigned "USA", IPROP is assigned "president" and iVAL is assigned "Reagan".

The equivalence transformations also use the pattern matcher. For instance, consider the following simple (but useful) transformation:

```
((iSI •)  (IW1  <NOMINAL>)  <POSS>
 (!W2  <NOMINAL>)  (!S2  •)  ?(IP  <PUNCT>))
::>
(NCONC  ISI 1W2 (LIST  'OF)  IW1  IS2  tP)
```

This transformation maps possessive constructions into attribute-value constructions, which we chose as canonical. For instance "Tell me about the VAX-785's performance." is mapped into "Tell

DYnamic PARsing is still in Its infant stages, requiring frequent changes in its software

[5]It was not our intention in building DYPAR that it outperform existing parsers in terms of theoiotic.il coverage. but rather that it replicate known peiformance in a moio natural, parsimonious, easier to extend manner However, our next step - integrating DYPAR and CASPAR strives for both impioved coverage and much more robust reliable performance in well defined domains

me about the performance of the VAX-785." The latter construction is recognized by a pattern-action rule. Since possessive constructions can occur in many contexts, the single transformation above saves us from duplicating pattern match rules for each different context where an attribute-value construction can occur.

3.2. A Sample DYPAR Dialogue

The control structure of DYPAR is an applicative condition-action cycle, which halts upon no rules being triggered (in which case a partial pattern-match strategy is attempted), or upon one of the pattern-action rules firing and passing its information to the data base manager. Let us see a dialogue fragment with DYPAR. For simplicity, we start out with an empty data base. Items in *italics* below were typed in by the user. Everything else was output by the system. Items in UPPER CASE are paraphrases of internal semantic network relations output by a rudimentary English generator. When DYPAR does not recognize concepts in the input, it prompts the user for additional information required by the integrity-checker in the data base manager.

```
*(parser)

You can build and query a simple semantic network
by stating information and questions in English.

>Fido is a nice dog.

Storing assertion in semantic net:   FIDO IS A DOG.

Inference:  DOG is a generic concept.

What is NICE?
>Nice is a disposition.

Storing assertion in semantic net:   NICE IS A DISPOSITION.

Is it correct to say that  THE DISPOSITION OF FIDO IS NICE?
 (Y,N): Yes.
Adding new assertion:   THE DISPOSITION OF FIDO IS NICE.

>Fido's cousin is Felix.

Transforming  (FIDO 'S COUSIN IS FELIX PERIOD)
into ::>      (THE COUSIN OF FIDO IS FELIX PERIOD)

Transforming  (THE COUSIN OF FIDO IS FELIX PERIOD)
into ::>      (FELIX IS THE COUSIN OF FIDO PERIOD)

Adding new assertion:   FELIX IS THE COUSIN OF FIDO.

>Cousin is a symmetric relation.

I already know that: COUSIN WAS A RELATION.

Adding new assertion:   THE RELATION OF COUSIN IS SYMMETRIC.
Inference: Adding new assertion:
   THE INVERSE OF COUSIN IS COUSIN.
Inference: Adding new assertion:
   FIDO IS THE COUSIN OF FELIX.
```

As we see in the above example, robust communication with the user requires not only a flexible domain-oriented parser, but also an interactive query capability and a natural language generator. However, the latter two processes are conceptually simpler, and not the topic of this paper.

4. Combining the Strengths of CASPAR and DYPAR

CASPAR illustrates the power of case-frame parsers when they can exploit very strong domain-specific semantic constraints. DYPAR illustrates the harmonious integration of three parsing strategies. However, these parsers are only first steps in exploiting the multi-strategy approach to develop real-world, robust, natural language interfaces. In terms of sophistication. DYPAR straddles the boundary between an advanced toy and a rudimentary real-applications system. One direction of continued development would be to enhance its pattern matcher, build additional general transformations, and augment its language interface to serve as a medium in which to express extensions to the grammar by a domain expert (not necessarily a natural language expert). A first step in the direction of automating and simplifying user extensibility has been taken in the Nano KLAUS system [13].

Our research is focused in a direction different from, but complementary to, the one taken in Nano KLAUS. Since the performance obtained by integrating several parsing strategies has, for both CASPAR and DYPAR, proven more effective than the application of any single strategy, we intend to extrapolate by including additional parsing strategies in future parsers. As a step along this road, we have designed a flexible control structure [6] that integrates case instantiation with other parsing strategies discussed in this paper, together with additional construction-specific strategies. As in CASPAR, the case oriented strategy is the dominant one. We expect this new design to provide a quantum jump in the range of applicability of our task-oriented parsers. Moreover, techniques such as expectation-driven disambiguation [2.19). developed in non-applied natural language work, can now be brought to bear in real-world applications. The reason why case frame parsers have not been developed in task-oriented domains is that, while they capture general principles admirably and can bring detailed semantic knowledge to bear, they are not well suited to recognizing specific idioms, compound nouns and the like. However, the addition of partial pattern matching (ideally suited to detecting idiomatic expressions) integrated with case frame instantiation and other parsing methods should provide a high degree of generality without sacrificing robustness.

Graceful interaction with the user is an important goal for any natural language front end whose users may be unfamiliar with computers (more details on our broader efforts towards this goal are given in [11,9]). People invariably produce ungrammatical utterances, leave out words, add interjections, and use terms outside the vocabulary of any system. It is essential that a real-world system "fail soft" in such circumstances, and interact with the user to enable graceful recovery. We saw a simple examples of this in DYPAR, and more mechanisms are discussed in [10J. The expectation setting provided by a case system incorporating domain knowledge may prove a more powerful tool to minimize failure than mechanisms based on relaxing grammatical rules or pattern matching requirements.

Consider, for instance, a file management system where a user may type

Transfer the flies in my directory to the accounts directory.

It is fairly clear to us humans that the user meant to type "files", even if we know perfectly well that "flies" is a legitimate word in our vocabulary.[6] A case frame system knows that the object case of the transfer command (as applied to the file-management domain) requires a logical data entity, which "flies" is not. Observing this violated semantic requirement, it can proceed to see whether by spelling correction, morphological decomposition,

[6] No presently implemented spelling correction scheme is ever applied to "correctly" spelled words.

or detecting potential omissions it can map "flies" into a known filler of that case. Here, spelling correction works (given a very restricted pool of candidate words satisfying the semantic requirements), and the system can proceed to inform the user of its correction (allowing the user to override if need be).

A striking advantage of our mixed-strategy approach is that the top-level case structure, in essence, partitions the semantic world into categories that can legitimately fill specific cases. The power of the approach derives from these top-down case-frame expectations significantly constraining bottom-up pattern matching. Thus, when a pattern matcher is invoked to parse the recipient case of a file transfer frame, it need only consider patterns (and semantic grammar constructs) that correspond to logical locations inside a computer. This form of expectation-driven parsing in restricted domains has two important advantages from the point of view of robustness:

- Many spurious parses are never generated (because patterns yielding potentially spurious matches are never tried in inappropriate contexts.)
- Additional knowledge, additional patterns, grammar rules, etc. can be added without a corresponding linear increase in parse time (since the case-frames focus only upon the relevant subset of patterns and rules). Thus, the efficiency of the system may actually increase with the addition of more domain knowledge (in effect enabling the case frames to restrict context even further). This behavior makes it possible to build the parser incrementally without the ever-present fear that a new extension may make the entire parser fail due to unexpected application of that extension in the wrong context.

We conclude this section by reiterating the central theme of our investigations: *Integration of multiple construction specific parsing strategies is a powerful organizing principle for robust, task-oriented natural language interfaces.*

5. Advantages for Data Base Interfaces

A further advantage of multiple construction-specific parsing techniques and grammar representations arises in the case of interfaces for accessing and updating data bases. Most current data base interfaces that use natural language are concerned purely with data base access rather than update, though see Kaplan [15] for some discussion of the problems involved in the latter activity. The typical approach of such interfaces has been to translate a users natural language questions about the contents of a data base into a formal query language which is then interpreted by a program specific to the particular data base to produce the required answer. The answer is then expressed to the user in some more or less natural format. The translation into the formal query language can be done directly from the input, as is the case in LADDER (20), or indirectly via a syntax tree of the input as in LUNAR [24].

While this arrangement is fine for pure data base access, it is less than optimal if a mixture of access and update is desired. The problem is that the structure of the query languages employed does not mirror the structure of the data base being queried. This means that an access request and an update request of essentially similar form will result in radically different parses. In turn, this means that constituents which are identical at the language level must be parsed in radically different ways depending on whether they are in an access or an update request.

To make this point concrete, consider the following two requests that might be presented to an interface to a data base concerning college courses:

Who is the instructor of Physics 247?
Change the instructor of Physics 247 to be Solway.

Translating the first input into a typical[7] formal query language might look something like:

```
(FOR (X in COURSES)
     (and (= (DEPT X) Economics)
          (= (NUMBER X) 247))
     (LIST (INSTRUC X)))
```

A corresponding translation for the update request[8] on the other hand might be:

```
(FOR (X in COURSES)
     (and (« (OEPT X) Economics)
          (= (NUMBER X) 247))
     (CHANGE-INSTRUC X Solway))
```

While the representation of "Economics 247" is the same in both cases, the treatment of "instructor" is quite different. In the access example, it is encoded into the access function, INSTRUC, and in the update example, it turns into the update function, CHANGE-INSTRUC. The reason for such radically different treatment is a desire to make the ouery language independent of the structure of the data. For example the relations between courses and their departments, numbers, and instructors could be contained in one file, or in three separate files. By making no assumption about the structure of the data, and just using neutral functions like INSTRUC and CHANG INSTRUC which assume that there is a relation between course and instructor, but nothing about the way it is represented, the query language avoids dependencies of this sort.

However, adopting a query language of this type has unfortunate consequences for a parser that must recognize both access and update requests. Clearly, it would be desirable for such a parser to use the same grammar rules to recognize "the instructor of Economics 247" in both examples above, but the target representation makes this quite inconvenient. Because "instructor" has to be translated in two quite different ways, it is most natural to control the parsing of the phrase from a higher level, so there would probably be rules[9] that recognized complete phrases like:

```
<tell-me> <instructor> of <course>
<change> <instructor> of <course> to <person>
```

These examples are actually patterned after, though not identical to, the language employed by Woods in the LUNAR system (24). which is one of the more easily human readable of the query languages that have been used Most DB query languages adopt a similar, if mote cryptic, formalism For instance. COOASYL *[7]* queries are stated in a Imeaiized form whose content resembles that of our example above Kaplan (14). however, first translates queries into a intermediate formalism, satisfying some but not all the features of the representation we propose in the following pages, before generating the cryptic COOASYL query form

8 We do not consider here "non obvious" interpretations for update requests like those discussed by Kaplan [15]

[9]The exact form of the grammar representation, network or pattern, is unimpoitant here, just the idea that phrases of this type would be recognized as a whole

and turned them into internal representations like the ones above. In order to be able to recognize patterns like "<instructor> of <course>" or better still "<slot> of <structure>" without regard to whether they were part of an access or update request, one would have to represent output from the parser something like:

```
[RequestType: Access
 SlotSpec:
   [Slot: Instructor
    Structure:
      [Type: COURSE Oept: Physics Number; 2471
```

```
[RequsstType: Update
 SlotSpec:
   [Slot: Instructor
    Structure:
      [Type: COURSE Oept: Physics Number: 247]

 UpdateValue: Solway
```

Here the representation of "the instructor of Economics 247" is the same for both the access and update requests.

This form of representation has the advantages we have described for parsing. A potential objection, however, is that the representation presumes too much about the structure of the data base. We do not believe that this objection is valid because the assumptions made are only about the logical structure of the data base from the user's point of view, rather than its detailed organization into files, etc Any relational data base would certainly fit into the slot and filler, property list, style of representation we have used, and other forms of logical organization could also be accommodated. In addition, the work in LADDER [20] shows how an abstract intermediate representation is very useful when several data bases of different detailed structure are involved, and the above style of representation should serve splendidly for that purpose.

So far, nothing we have said in this section makes the construction-specific approach to parsing preferable to the uniform approach, with respect to data base interactions. There would be nothing to stop a uniform parser producing the kind of representation we are advocating for update and access requests just as well as a construction-specific one could. The primary advantage of construction-specific parsing here involves describing the relation between the logical structure of the data base and the input language. As we noted in the introduction, since the specific constructions dealt with can be those "natural" for the task domain, when using a construction-specific approach, the input language can be defined in a way that is tied closely to the slot and filler structure of the objects of the data base. To be more precise, the input syntax for the description of a given kind of data base object can be expressed in terms of the fillers for slots of that object type embedded in whatever kind of construction is natural for describing such an object. As an example of the kind of language definition we have in mind, consider the following three complementary syntax definitions for descriptions of a course.

```
[SyntaxType: NounWithCases
 Head: (course section class)
 PostCases: [Instructor:
               (by (taught by) (given by))
             Department: ((held In))
             Number: (numbered)
```

```
[SyntaxType: PossessiveCases
 NamedCases:
     [Instructor:
        (instructor teacher (faculty member))
      Department: (department)
```

```
[SyntaxType: SlotPattern
 Pattern: (Department Number)
```

Without going into great detail, the first syntax definition above says that a construction type of NOUNWITHCASES may be used to describe a COURSE. This means that it may be described by a head noun (a list of alternatives is given), optionally preceded by a determiner (part of the construction definition), followed by a sequence of descriptive cases. These cases are defined as a property list indicated by POSTCASES. The indicators (INSTRUCTOR, DEPARTMENT, etc) are slot names of the object type, and the values are a list of possible case markers for that case (both single words and linear patterns of words). For instance, "taught by" can signal the presence of an instructor description. The fact that INSTRUCTOR is a slot of COURSE, and the syntax for fillers of that slot are recorded elsewhere. The second syntax definition gives a listing of the slots of COURSE, and the words used to describe them. From this information, a phrase like "the instructor of <course>" can be parsed. Finally, an alternative syntax for COURSE of (<department> <number>) is given. We have already developed a similar kind of language definition formalism for command interface applications [1] We intend to extend this work to interfaces that access and update data bases. For instance, the noun-phrase head and the case markers can be represented by more complex patterns that may include non-terminals from a semantic grammar •• similar to the mechanism used (in a less discriminating fashion) in the DYPAR system.

6. Conclusion

The goal of this paper has been to explain the advantages of a multi strategy, construction-specific approach to parsing in applied natural language processing, largely through an examination of two "toy" parsers that we have constructed. The construction-specific multi strategy approach exploits prior knowledge of expected constructions, domain semantics, and strategies optimized to recognize each construction type. The two parsers demonstrated clear advantages for the construction-specific approach in parsing both grammatical and ungrammatical input. We intend to construct a single, practical, multi strategy parsing system that combines the best aspects of both simpler systems in a more complex, embedded constituent control structure. Finally, we discussed some issues in data base access and update, and showed that a construction-specific approach, coupled with a case-structured data base description, offers a promising approach to a unified, interactive data base query and update system.

References

Ball, J. E. and Hayes, P. J., "Representation of Task-Independent Knowledge in a Gracefully Interacting User Interface," *Proc. 1st Annual Meeting of the American Association for Artificial Intelligence,* Stanford University, August 1980, pp. 116 120.

Birnbaum, L. and Selfridge. M., "Conceptual Analysis in Natural Language," in *Inside Computer Understanding,* R. Schank and C Riesbeck, eds., New Jersey: Erlbaum Assoc, 1980, pp. 318-353.

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

Burton, R. R., "Semantic Grammar: An Engineering Technique for Constructing Natural Language Understanding Systems." Tech. report 3453, Bolt Beranek and Newman, 1975.

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

Carbonell, J. G. and Hayes. P. J., "Dynamic Strategy Selection in Flexible Parsing," *Proc. of J9th Annual Meeting of the Assoc, for Comput. Ling.,* Stanford University, June 1981 .

ACM Publications, *Data Base Task Group of CODASYL Programming Language Committee Report.* NY, 1971.

Hayes P. J., "Focused Interaction in Flexible Parsing," Proc. *of 19th Annual Meeting of the Assoc, for Comput. Ling.,* Stanford University, June 1981 .

Hayes, P. J., Ball, J. E., and Reddy, R., "Breaking the Man-Machine Communication Barrier,*"Computer,* March 1981 .

Hayes, P. J. and Mouradian, G. V., "Flexible Parsing," Proc. *of 18th Annual Meeting of the Assoc, for Comput. Ling.,* Philadelphia, June 1980 , pp. 97-103.

Hayes, P. J., and Reddy, R., "An Anatomy of Graceful Interaction in Man-Machine Communication," Tech. report, Computer Science Department, Carnegie Mellon University, 1979,.

Hendrix, G. G., "Human Engineering for Applied Natural Language Processing," *Proc. Fifth Int. Jt. Conf. on Artificial Intelligence,* 1977 , pp. 183-191.

Hendrix, G. G. and Haas, N., "Acquiring Knowledge for Information Management," in *Machine Learning,* R. S. Michalski, J. G. Carbonell and T. M. Mitchell, eds., Palo Alto, CA: Tioga Pub. Co., 1981.

Kaplan, S. J., *Cooperative Responses from a Portable Natural Language Data Base Query System,* PhD dissertation, Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia, 1979.

15. Kaplan, S. J., "Interpreting Natural Language Data Base Updates," *Submitted to the 19th Annual Meeting of the Assoc, for Comput. Ling.,* Stanford University, June 1981 .

16. Kwasny, S. C. and Sondheimer, N. K., "Ungrammatically and Extra-Grammatically in Natural Language Understanding Systems," Proc. *of 17th Annual Meeting of the Assoc, for Comput. Ling.,* La Jolla, Ca., August 1979 , pp. 19 23.

17. Marcus, M. A., *A Theory of Syntactic Recognition for Natural Language,* MIT Press, Cambridge, Mass., 1980.

18. Parkison, R.C., Colby, K. M., and Faught, W. S., "Conversational Language Comprehension Using Integrated Pattern-Matching and Parsing ," *Artificial Intelligence,* Vol. 9, 1977 , pp. 111-134.

19. Riesbeck, C. and Schank, R. C, "Comprehension by Computer: Expectation Based Analysis of Sentences in Context," Tech. report 78, Computer Science Department, Yale University, 1976.

20. Sacerdoti, E D., "Language Access to Distributed Data with Error Recovery," *Proc. Fifth Int. Jt. Conf. on Artificial Intelligence,* 1977 , pp. 196-202.

21. Waltz, D.L., "An English Language Question Answering System for a Large Relational Data Base," *CACM,* Vol. 21, No. 7, 1978, pp. 526539.

22. Weischedel, R. M. and Black, J., "Responding to Potentially Unparseable Sentences," Tech. report 79/3, Dept. of Computer and Information Sciences, University of Delaware, 1979.

23. Wilks, Y. A., "Preference Semantics," in *Formal Semantics of Natural Language,* Keenan, ed., Cambridge University Press, 1975.

24. Woods, W. A , Kaplan, R. M , and Nash Webber, B., "The Lunar Sciences Language System: Final Report," Tech. report 2378, Bolt, Beranek, and Newman, Inc., Cambridge, Mass., 1972.