Dynamic Strategy Selection in Flexible Parsing

Jaime G. Carbonell and Philip J. Hayes
Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract

Robust natural language interpretation requires strong semantic domain models, "fail-soft" recovery heuristics, and very flexible control structures. Although single-strategy parsers have met with a measure of success, a multi-strategy approach is shown to provide a much higher degree of flexibility, redundancy, and ability to bring task-specific domain knowledge (in addition to general linguistic knowledge) to bear on both grammatical and ungrammatical input. A parsing algorithm is presented that integrates several different parsing strategies, with case-frame instantiation dominating. Each of these parsing strategies exploits different types of knowledge; and their combination provides a strong framework in which to process conjunctions, fragmentary input, and ungrammatical structures, as well as less exotic, grammatically correct input. Several specific heuristics for handling ungrammatical input are presented within this multi-strategy framework.

1. Introduction

When people use language spontaneously, they often do not respect grammatical niceties. Instead of producing sequences of grammatically well-formed and complete sentences, they often miss out or repeat words or phrases, break off what they are saying and rephrase or replace it, speak in fragments, or use otherwise incorrect grammar. While other people generally have little trouble comprehending ungrammatical utterances, most natural language computer systems are unable to process errorful input at all. Such inflexibility in parsing is a serious impediment to the use of natural language in interactive computer systems. Accordingly, we [6] and other researchers including Weischedel and Black [14], and Kwasny and Sondheimer [9], have attempted to produce flexible parsers, i.e. parsers that can accept ungrammatical input, correcting the errors when possible, and generating several alternative interpretations if appropriate.

While different in many ways, all these approaches to flexible parsing operate by applying a uniform parsing process to a uniformly represented grammar. Because of the linguistic performance problems involved, this uniform procedure cannot be as simple and elegant as the procedures followed by parsers based on a pure linguistic competence model, such as Parsifal [10]. Indeed, their parsing procedures may involve several strategies that are applied in a predetermined order when the input deviates from the grammar, but the choice of strategy never depends on the specific type of construction being parsed. In light of experience with our own flexible parser, we have come to believe that such uniformity is not conducive to good flexible parsing. Rather, the strategies used should be dynamically selected according to the type of construction being parsed. For instance, partial linear pattern matching may be well suited to the flexible parsing of idiomatic phrases, or specialized noun phrases such as names, dates, or addresses (see also [5]), but case constructions, such as noun phrases with trailing prepositional phrases, or imperative phrases, require case-oriented parsing strategies. The underlying principle is simple: ***The appropriate knowledge must be brought to bear at the right time — and it must not interfere at other times.*** Though the initial motivation for this approach sprang from the needs of flexible parsing, such construction-specific techniques can provide important benefits even when no grammatical deviations are encountered, as we will show. This observation may be related to the current absence of any single universal parsing strategy capable of exploiting all knowledge sources (although ELI [12] and its offspring [2] are efforts in this direction).

Our objective here is not to create the ultimate parser, but to build a very flexible and robust task-oriented parser capable of exploiting all relevant domain knowledge as well as more general syntax and semantics. The initial application domain for the parser is the central component of an interface to various computer subsystems (or tools). This interface and, therefore the parser, should be adaptable to new tools by substituting domain-specific data bases (called "tool descriptions") that govern the behavior of the interface, including the invocation of parsing strategies, dictionaries and concepts, rather than requiring any domain adaptations by the interface system itself.

With these goals in mind, we proceed to give details of the kinds of difficulties that a uniform parsing strategy can lead to, and show how dynamically-selected construction-specific techniques can help. We list a number of such specific strategies, then we focus on our initial implementation of two of these strategies and the mechanism that dynamically selects between them while parsing task-oriented natural language imperative constructions. Imperatives were chosen largely because commands and queries given to a task-oriented natural language front end often take that form [6].

2. Problems with a Uniform Parsing Strategy

Our present flexible parser, which we call FlexP, is intended to parse correctly input that corresponds to a fixed grammar, and also to deal with input that deviates from that grammar by erring along certain classes of common ungrammaticalities. Because of these goals, the parser is based on the combination of two uniform parsing strategies: bottom-up parsing and pattern-matching. The choice of a bottom-up rather than a top-down strategy was based on our need to recognize isolated sentence fragments, rather than complete sentences, and to detect restarts and continuations after interjections. However, since completely bottom-up strategies lead to the consideration of an unnecessary number of alternatives in correct input, the algorithm used allowed some of the economies of top-down parsing for non-deviant input. Technically speaking, this made the parser *left-corner* rather than bottom-up. We chose to use a grammar of linear patterns rather than, say, a transition network because pattern-matching meshes well with bottom-up parsing by allowing lookup of a pattern from the presence in the input of any of its constituents; because pattern-matching facilitates recognition of utterances with omissions and substitutions when patterns are recognized on the basis of partial matches; and because pattern-matching is necessary for the recognition of idiomatic phrases. More details of the justifications for these choices can be found in [6].

FlexP has been tested extensively in conjunction with a gracefully interacting interface to an electronic mail system [1]. "Gracefully interacting" means that the interface appears friendly, supportive, and robust to its user. In particular, graceful interaction requires the system to tolerate minor input errors and typos, so a flexible parser is an important component of such an interface. While FlexP performed this task adequately, the experience turned up some problems related to the

[1]This research was sponsored in part 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 in part by the Air Force Office of Scientific Research under Contract F49620-79-C-0143. 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 DARPA, the Air Force Office of Scientific Research or the US government.

major theme of this paper. These problems are all derived from the incompatibility between the uniform nature of the grammar representation and the kinds of flexible parsing strategies required to deal with the inherently non-uniform nature of some language constructions. In particular:

- Different elements in the pattern of a single grammar rule can serve radically different functions and/or exhibit different ease of recognition. Hence, an efficient parsing strategy should react to their apparent absence, for instance, in quite different ways.

- The representation of a single unified construction at the language level may require several linear patterns at the grammar level, making it impossible to treat that construction with the integrity required for adequate flexible parsing.

The second problem is directly related to the use of a pattern-matching grammar, but the first would arise with any uniformly represented grammar applied by a uniform parsing strategy.

For our application, these problems manifested themselves most markedly by the presence of case constructions in the input language. Thus, our examples and solution methods will be in terms of integrating case-frame instantiation with other parsing strategies. Consider, for example, the following noun phrase with a typical postnominal case frame:

"the messages from Smith about ADA pragmas dated later than Saturday".

The phrase has three cases marked by "from", "about", and "dated later than". This type of phrase is actually used in FlexP's current grammar, and the basic pattern used to recognize descriptions of messages is:

```
<?determiner *MessageAdj MessageHead *MessageCase>
```

which says that a message description is an optional (?) determiner, followed by an arbitrary number (*) of message adjectives followed by a message head word (i.e. a word meaning "message"), followed by an arbitrary number of message cases. In the example, "the" is the determiner, there are no message adjectives, "messages" is the message head word, and there are three message cases: "from Smith", "about ADA pragmas", and "dated later than". Because each case has more than one component, each must be recognized by a separate pattern:

```
<%from Person>
<%about Subject>
<%since Date>
```

Here % means anything in the same word class, "dated later than", for instance, is equivalent to "since" for this purpose.

These patterns for message descriptions illustrate the two problems mentioned above: the elements of the case patterns have radically different functions - the first elements are case markers, and the second elements are the actual subconcepts for the case. Since case indicators are typically much more restricted in expression, and therefore much easier to recognize than their corresponding subconcepts, a plausible strategy for a parser that "knows" about case constructions is to scan input for the case indicators, and then parse the associated subconcepts top-down. This strategy is particularly valuable if one of the subconcepts is malformed or of uncertain form, such as the subject case in our example. Neither "ADA" nor "pragmas" is likely to be in the vocabulary of our system, so the only way the end of the subject field can be detected is by the presence of the case indicator "from" which follows it. However, the present parser cannot distinguish case indicators from case fillers - both are just elements in a pattern with exactly the same computational status, and hence it cannot use this strategy.

The next section describes an algorithm for flexibly parsing case constructions. At the moment, the algorithm works only on a mixture of case constructions and linear patterns, but eventually we envisage a number of specific parsing algorithms, one for each of a number of construction types, all working together to provide a more complete flexible parser.

Below, we list a number of the parsing strategies that we envisage might be used. Most of these strategies exploit the constrained task-oriented nature of the input language:

- **Case-Frame Instantiation** is necessary to parse general imperative constructs and noun phrases with postnominal modifiers. This method has been applied before with some success to linguistic or conceptual cases [12] in more general parsing tasks. However, it becomes much more powerful and robust if domain-dependent constraints among the cases can be exploited. For instance, in a file-management system, the command "Transfer UPDATE.FOR to the accounts directory" can be easily parsed if the information in the unmarked case of *transfer* ("update.for" in our example) is parsed by a file-name expert, and the destination case (flagged by "to") is parsed not as a physical location, but a logical entity inside a machine. The latter constraint enables one to interpret "directory" not as a phone book or bureaucratic agency, but as a reasonable destination for a file in a computer.

- **Semantic Grammars** [8] prove useful when there are ways of hierarchically clustering domain concepts into functionally useful categories for user interaction. Semantic grammars, like case systems, can bring domain knowledge to bear in disambiguating word meanings. However, the central problem of semantic grammars is non-transferability to other domains, stemming from the specificity of the semantic categorization hierarchy built into the grammar rules. This problem is somewhat ameliorated if this technique is applied only to parsing selected individual phrases [13], rather than being responsible for the entire parse. Individual constituents, such as those recognizing the initial segment of factual queries, apply in may domains, whereas a constituent recognizing a clause about file transfer is totally domain specific. Of course, this restriction calls for a different parsing strategy at the clause and sentence level.

- **(Partial) Pattern Matching** on strings, using non-terminal semantic-grammar constituents in the patterns, proves to be an interesting generalization of semantic grammars. This method is particularly useful when the patterns and semantic grammar non-terminal nodes interleave in a hierarchical fashion.

- **Transformations to Canonical Form** prove useful both for domain-dependent and domain-independent constructs. For instance, the following rule transforms possessives into "of" phrases, which we chose as canonical:

```
[<ATTRIBUTE> in possessive form,
 <VALUE> legitimate for attribute]
                =>
[<VALUE> "OF" <ATTRIBUTE> in simple form]
```

  Hence, the parser need only consider "of" constructions ("file's destination" => "destination of file"). These transforms simplify the pattern matcher and semantic grammar application process, especially when transformed constructions occur in many different contexts. A rudimentary form of string transformation was present in PARRY [11].

- **Target-specific methods** may be invoked to parse portions of sentences not easily handled by the more general methods. For instance, if a case-grammar determines that the case just signaled is a proper name, a special name-expert strategy may be called. This expert knows that names

can contain unknown words (e.g., Mr. Joe Gallen D'Aguila is obviously a name with D'Aguila as the surname) but subject to ordering constraints and morphological preferences. When unknown words are encountered in other positions in a sentence, the parser may try morphological decomposition, spelling correction, querying the user, or more complex processes to induce the probable meaning of unknown words, such as the project-and-integrate technique described in [3]. Clearly these unknown-word strategies ought to be suppressed in parsing person names.

3. A Case-Oriented Parsing Strategy

As part of our investigations in task-oriented parsing, we have implemented (in addition to FlexP) a pure case-frame parser exploiting domain-specific case constraints stored in a declarative data structure, and a combination pattern-match, semantic grammar, canonical-transform parser. All three parsers have exhibited a measure of success, but more interestingly, the strengths of one method appear to overlap with the weaknesses of a different method. Hence, we are working towards a single parser that dynamically selects its parsing strategy to suit the task demands.

Our new parser is designed primarily for task domains where the prevalent forms of user input are commands and queries, both expressed in imperative or pseudo-imperative constructs. Since in imperative constructs the initial word (or phrase), establishes the case-frame for the entire utterance, we chose the case-frame parsing strategy as primary. In order to recognize an imperative command, and to instantiate each case, other parsing strategies are invoked. Since the parser knows what can fill a particular case, it can choose the parsing strategy best suited for linguistic constructions expressing that type of information. Moreover, it can pass any global constraints from the case frame or from other instantiated cases to the subsidiary parsers - thus reducing potential ambiguity, speeding the parse, and enhancing robustness.

Consider our multi-strategy parsing algorithm as described below. Input is assumed to be in the imperative form:

1. *Apply string PATTERN-MATCH to the initial segment of the input using only the patterns previously indexed as corresponding to command words/phrases in imperative constructions.* Patterns contain both optional constituents and non-terminal symbols that expand according to a semantic grammar. (E.g., "copy" and "do a file transfer" are synonyms for the same command in a file management system.)

2. *Access the CASE-FRAME associated with the command just recognized, and push it onto the context stack.* In the above example, the case-frame is indexed under the token <COPY>, which was output by the pattern matcher. The case frame consists of list of pairs ([case-marker] [case-filler-information], ...).

3. *Match the input with the case markers using the PATTERN-MATCH system described above.* If no match occurs, assume the input corresponds to the unmarked case (or the first unmarked case, if more than one is present), and proceed to the next step.

4. *Apply the parsing strategy indicated by the type of construct expected as a case filler. Pass any available case constraints to the sub-parser.* A partial list of parsing strategies indicated by expected fillers is:

   - **Sub-imperative** -- Case-frame parser, starting with the command-identification pattern match above.
   - **Structured-object** (e.g., a concept with subattributes) -- Case-frame parser, starting with the pattern-matcher invoked on the list of patterns corresponding to the names (or compound names) of the semantically permissible structured objects, followed by case-frame parsing of any present subattributes.
   - **Simple Object** -- Apply the pattern matcher, using only the patterns indexed as relevant in the case-filler-information field.
   - **Special Object** -- Apply the parsing strategy applicable to that type of special object (e.g., proper names, dates, quoted strings, stylized technical jargon, etc...)
   - **None of the above** -- (Errorful input or parser deficiency) Apply the graceful recovery techniques discussed below.

5. *If an embedded case frame is activated, push it onto the context stack.*

6. *When a case filler is instantiated, remove the <case-marker>, <case-filler-information> pair from the list of active cases in the appropriate case frame, proceed to the next case-marker, and repeat the process above until the input terminates.*

7. *If all the cases in a case frame have been instantiated, pop the context stack until that case frame is no longer in it.* (Completed frames typically reside at the top of the stack.)

8. *If there is more than one case frame on the stack when trying to parse additional input, apply the following procedure:*

   - If the input only matches a case marker in one frame, proceed to instantiate the corresponding case-filler as outlined above. Also, if the matched case marker is not on the most embedded case frame (i.e., at the top of the context stack), pop the stack until the frame whose case marker was matched appears at the top of the stack.
   - If no case markers are matched, attempt to parse unmarked cases, starting with the most deeply embedded case frame (the top of the context stack) and proceeding outwards. If one is matched, pop the context stack until the corresponding case frame is at the top. Then, instantiate the case filler, remove the case from the active case frame, and proceed to parse additional input. If more than one unmarked case matches the input, choose the most embedded one (i.e., the most recent context) and save the state of the parse on the global history stack. (This suggests an ambiguity that cannot be resolved with the information at hand.)
   - If the input matches more than one case marker in the context stack, try to parse the case filler via the indexed parsing strategy for each filler-information slot corresponding to a matched case marker. If more than one case filler parses (this is somewhat rare situation - indicating underconstrained case frames or truly ambiguous input) save the state in the global history stack and pursue the parse assuming the most deeply embeded constituent. [Our case-frame attachment heuristic favors the most local attachment permitted by semantic case constraints.]

9. *If a conjunction or disjunction occurs in the input, cycle through the context stack trying to parse the right-hand side of the conjunction as filling the same case as the left hand side. If no such parse is feasible, interpret the conjunction as top-level, e.g, as two instances of the same imperative, or two different imperatives. If more than one parse results, interact with the user to disambiguate.* To illustrate this simple process, consider:

   "Transfer the programs written by Smith and Jones to ..."

   "Transfer the programs written in Fortran and the census data files to ..."

   "Transfer the programs written in Fortran and delete ..."

   The scope of the first conjunction is the "author" subattribute of program, whereas the scope of the second conjunction is the unmarked "object" case of the transfer action. Domain knowledge in the case-filler information of the "object" case in the "transfer" imperative inhibits "Jones" from matching a potential object for electronic file transfer. Similarly "Census data files" are inhibited from matching the "author" subattribute of a program. Thus conjunctions in the two syntactically comparable examples are scoped differently by our semantic-scoping rule relying on domain-specific case information. "Delete" matches no active case filler, and hence it is parsed as the initial segment of a second conjoined utterance. Since "delete" is a known imperative, this parse succeeds.

10. *If the parser fails to parse additional input, pop the global history stack and pursue an alternate parse. If the stack is empty, invoke the graceful recovery heuristics.* Here the DELTA-MIN method [4] can be applied to improve upon depth-first unwinding of the stack in the backtracking process.

11. *If the end of the input is reached, and the global history stack is not empty, pursue the alternate parses. If any survive to the end of the input (this should not be the case unless true ambiguity exists), interact with the user to select the appropriate parse (see [7]).*]

The need for embeded case structures and ambiguity resolution based on domain-dependent semantic expectations of the case fillers is illustrated by the following pair of sentences:

"Edit the programs in Fortran"
"Edit the programs in Teco"

"Fortran" fills the *language* attribute of "program", but cannot fill either the *location* or *instrument* case of Edit (both of which can be signaled by "in"). In the second sentence, however, "Teco" fills the *instrument* case of the verb "edit" and none of the attributes of "program". This disambiguation is significant because in the first example the user specified ***which*** programs (s)he wants to edit, whereas in the second example (s)he specified ***how*** (s)he wants to edit them.

The algorithm presented is sufficient to parse grammatical input. In addition, since it operates in a manner specifically tailored to case constructions, it is easy to add modifications dealing with deviant input. Currently, the algorithm includes the following steps that deal with ungrammaticality:

12. *If step 4 fails, i.e. a filler of appropriate type cannot be parsed at that position in the input, then repeat step 3 at successive points in the input until it produces a match, and continue the regular algorithm from there. Save all words not matched on a SKIPPED list.* This step takes advantage of the fact that case markers are often much easier to recognize than case fillers to realign the parser if it gets out of step with the input (because of unexpected interjections, or other spurious or missing words).

13. *If words are on SKIPPED at the end of the parse, and cases remain unfilled in the case frames that were on the context stack at the time the words were skipped, then try to parse each of the case fillers against successive positions of the skipped sequences.* This step picks up cases for which the marker was incorrect or garbled.

14. *If words are still on SKIPPED attempt the same matches, but relax the pattern matching procedures involved.*

15. *If this still does not account for all the input, interact with the user by asking questions focussed on the uninterpreted part of the input.* The same focussed interaction technique (discussed in [7]) is used to resolve semantic ambiguities in the input.

16. *If user interaction proves impractical, apply the project-and-integrate method [3] to narrow down the meanings of unknown words by exploiting syntactic, semantic and contextual cues.*

These flexible parsing steps rely on the construction-specific aspects of the basic algorithm, and would not be easy to emulate in either a syntactic ATN parser or one based on a pure semantic grammar.

A further advantage of our mixed-strategy approach is that the top-level case structure, in essence, partitions the semantic world dynamically into categories according to the semantic constraints on the active case fillers. Thus, when a pattern matcher is invoked to parse the recipient case of a file-transfer case 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 adds a two-fold effect to its robustness:

- Many spurious parses are never generated (because patterns yielding potentially spurious matches are never tried in inappropriate contexts.)

- Additional knowledge (such as 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 further restrict context). This behavior makes it possible to incrementally build the parser without the ever-present fear that a new extension may make the entire parser fail due to an unexpected application of that extension in the wrong context.

In closing, we note that the algorithm described above does not mention interaction with morphological decomposition or spelling correction. Lexical processing is particularly important for robust parsing; indeed, based on our limited experience, lexical-level errors are a significant source of deviant input. The recognition and handling of lexical-deviation phenomena, such as abbreviations and misspellings, must be integrated with the more usual morphological analysis. Some of these topics are discussed independently in [6]. However, integrating resilient morphological analysis with the algorithm we have outlined is a problem we consider very important and urgent if we are to construct a practical flexible parser.

4. Conclusion

To summarize, uniform parsing procedures applied to uniform grammars are less than adequate for parsing ungrammatical input. As our experience with such an approach shows, the uniform methods are unable to take full advantage of domain knowledge, differing structural roles (e.g., case markers and case fillers), and relative ease of identification among the various constituents in different types of

constructions. Instead, we advocate integrating a number of different parsing strategies tailored to each type of construction as dictated by the application domain. The parser should dynamically select parsing strategies according to what type of construction it expects in the course of the parse. We described a simple algorithm designed along these lines that makes dynamic choices between two parsing strategies, one designed for case constructions and the other for linear patterns. While this dynamic selection approach was suggested by the needs of flexible parsing, it also seemed to give our trial implementation significant efficiency advantages over single-strategy approaches for grammatical input.

5. References

1. 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, American Assoc. for Artificial Intelligence, Stanford University, August, 1980, pp. 116-120.

2. 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.

3. Carbonell, J. G. Towards a Self-Extending Parser. Proceedings of the 17th Meeting of the Association for Computational Linguistics, ACL-79, 1979, pp. 3-7.

4. Carbonell, J. G. Δ-MIN: A Search-Control Method for Information-Gathering Problems. Proceedings of the First AAAI Conference, AAAI-80, August, 1980.

5. Gershman, A. V. *Knowledge-Based Parsing*. Ph.D. Th., Yale University, April 1979. Computer Sci. Dept. report # 156

6. 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.

7. Hayes P. J. Focused Interaction in Flexible Parsing. Carnegie-Mellon University Computer Science Department, 1981.

8. Hendrix, G. G., Sacerdoti, E. D. and Slocum, J. Developing a Natural Language Interface to Complex Data. Tech. Rept. Artificial Intelligence Center., SRI International, 1976.

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

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

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

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

13. Waltz, D. L. and Goodman, A. B. Writing a Natural Language Data Base System. IJCAIVproc, IJCAI-77, 1977, pp. 144-150.

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