UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 81-0852	2. GOVT ACCESSION NO. AD-A109076	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) FLEXIBLE PARSING (5)	5. TYPE OF REPORT & PERIOD COVERED Annual 07-01-80 to 06-30-81
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Philip J. Hayes, Raj Reddy	8. CONTRACT OR GRANT NUMBER(s) F49620-79-C-0143
9. PERFORMING ORGANIZATION NAME AND ADDRESS Carnegie-Mellon University, 5000 Forbes Ave., Pittsburgh, Pa. 15213	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F; 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS Directorate of Mathematical & Information Sciences, Air Force Office of Scientific Research, Bolling AFB DC 20332	12. REPORT DATE October 12, 1981
	13. NUMBER OF PAGES 13
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) LEVEL II	15. SECURITY CLASS. (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

(12) 16

DTIC ELECTE DEC 30 1981 H

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

applied natural language, flexible parsing, bottom-up parsing, friendly interfaces, construction-specific parsing, multi-strategy parsing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The work reported falls into two distinct phases. In the first phase, the implementation of the FlexP parser was completed according to the design described in last year's annual report, and was evaluated through use in a gracefully interacting interface. This application showed: that FlexP was able to parse both grammatical and ungrammatical imput according to a simple grammar of pattern-matching rewrite rules, that the bottom-up approach of FlexP was helpful in the case of ungrammatical imput, and that a grammar suitable for use by FlexP could be defined in terms natural to the domain of interaction

DD FORM 1 JAN 73 1473

UNCLASSIFIED 403 530
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD A109076
DTIC FILE COPY

ITEM #20, CONTINUED:

of the interface. However, the experimental use of FlexP also made it clear that FlexP had certain problems, largely due to the uniform nature of its grammar.

In the second phase, the set of design choices on which FlexP was based was reviewed to determine if it was possible to resolve the problems that FlexP had without also losing the desirable aspects of its performance. This review led to the replacement of a single parsing strategy based on linear pattern matching with an approach based on multiple construction-specific parsing strategies. This approach was evaluated in a preliminary way through the construction of two small parsers, CASPAR and DYPAR, and was found to show considerable promise. Accordingly, design based on the same approach was begun for a more complete parser for restricted domain natural languages, and for a parser oriented towards an artificial command languate of the type common in interactive user interfaces.

Accession For
NTIS GRA&I
DTIC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist Special
A

Table of Contents

Approved for public release;
distribution unlimited.

81 12 29 040

1. Research Objectives

1. When people use language spontaneously, they often do not adhere strictly to commonly accepted standards of grammaticality. The primary objective of this project is to develop flexible computer parsing techniques which can deal with the various kinds of ungrammaticalities that arise, both on the lexical and the phrase level. The kinds of ungrammaticality we wish to deal with include at the lexical level:

   - misspelt words
   - novel words whose role can be inferred from context
   - erroneous segmentation between words (arising from the omission of spaces, or the inclusion of spurious spaces or punctuation)
   - lexical items which are entered in one form and then changed to another

   and at the phrase level:

   - input which is broken off and then restarted
   - interjected words and phrases
   - omitted or substituted words and phrases
   - fragmentary or otherwise elliptical input
   - agreement failure
   - idioms

2. The design space for parsers is very large. We aim to develop a set of design choices which will result in parsers well suited to our primary goal. The design choices we are currently using are listed below. The second one is different from the corresponding choice in our original proposal and previous report. The difference reflects our experience with the earlier choice as explained in the body of this report.

   - bottom-up rather than top-down parsing, except in certain situations in which top-down prediction is highly constraining
   - use of several different parsing strategies, each tailored to a particular type of construction, and selected between on a dynamic basis
   - provision for the suspension and later resumption of a partial parse at a non-adjacent part of the input string

3. We intend to develop flexible parsing techniques in the context of interfaces to interactive computer systems. We are working with two types of interface language:

   a. limited-domain natural languages, i.e. languages with the syntax of (possibly a

MATTHEW J.
Chief, Technical Information Division

subset of natural language, but whose semantics are limited to those of the interactive system being interfaced to.

b. more restrictive artificial languages of the sort currently found in computer interfaces

Later, we intend to investigate how easily the techniques developed for these kinds of languages can be transferred to more general natural language.

4. We intend to investigate formalisms for specifying domain-dependent grammars in a convenient way for both of the types of language mentioned above.

2. Status of the Research Effort

2.1. Overview

Our work during the past year fell into two distinct phases. In the first part of the year, we completed the implementation of the FlexP parser according to the design described in last year's annual report, and evaluated it through use in a gracefully interacting interface developed under other support. This application showed:

- that FlexP was able to parse both grammatical and ungrammatical input according to a simple grammar of pattern-matching rewrite rules,
- that the bottom-up approach of FlexP was helpful in the case of ungrammatical input,
- and that a grammar suitable for use by FlexP could be defined in terms natural to the domain of interaction of the interface.

However, the experimental use of FlexP also made it clear that FlexP had certain problems, largely due to the uniform nature of its grammar. These problems caused FlexP to parse some ungrammatical input inefficiently, and in other cases to generate an unnecessarily large number of alternative interpretations of ungrammatical input.

In the second part of the year, based on the experience gained through the implementation and use of FlexP, we reviewed the set of design choices on which FlexP was based to determine if it was possible to resolve the problems that FlexP had without also losing the desirable aspects of its performance. This review led us to replace the design choice of a single parsing strategy based on linear pattern matching with an approach based on multiple parsing strategies, one for each construction class of the language being parsed, with the strategies being selected between on a dynamic basis. This multi-strategy, construction-specific approach was evaluated in a preliminary way through the construction of two small parsers, CASPAR and DYPAR, and was found to show

considerable promise. We intend to continue with this approach through the construction of a larger, more complete, parser based on the same approach.

In this second phase of the year's work, we also applied the multi-strategy, construction-specific approach to parsing the input for a revised gracefully interacting interface being constructed under the same funding as the one mentioned above. This work has led to the design of a parser based on the same principles as CASPAR and DYPAR, but oriented towards an artificial command language of the type common in interactive user interfaces, rather than restricted domain natural languages of the type CASPAR, DYPAR, and FlexP are designed to parse.

2.2. Completion and Evaluation of FlexP

The design for FlexP and its rationale were described in last year's annual report and in more detail in[6]. The implementation that we completed in the first phase of this year's work was in all important respects faithful to that design. Therefore, to avoid redundancy, this report will only summarize FlexP and concentrate instead on its evaluation.

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 derived from 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 potentially spurious 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 for three reasons: 1) 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. 2) Pattern-matching facilitates recognition of utterances with omissions and substitutions when patterns are recognized on the basis of partial matches. 3) 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, problems of efficiency and of unnecessary ambiguity showed up through this experimentation - examples are given below. The problems that arose are rooted in the incompatibility between the uniform nature of the pattern-matching rewrite rule grammar representation used by FlexP 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 example application, these problems manifested themselves most markedly by the presence of case constructions in the input language. 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. Also, a single construction at the language level is spread over several patterns in the grammar. This has two undesirable consequences for the parsing process - inefficiency and the generation of unnecessary ambiguities.

First, let us examine how inefficiency arises. Because the parser has no information about the relationship between the cases and the top-level pattern (other than that the results of the case patterns match the last element in the top-level pattern), several powerful, but specialized, strategies for dealing with (regular or irregular) case constructions cannot be employed with a resulting loss of parsing efficiency. For instance, since case indicators are typically much more restricted in range of 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, FlexP cannot distinguish case indicators from case fillers - both are just elements in a pattern with exactly the same computational status. Hence it cannot use this strategy and inefficiency results.

The second consequence of the general problems mentioned above is the generation of unnecessary alternative parses. For instance, if an object type can appear in more than one slot of a case frame, and a case indicator for such an object is omitted on input, then a parser dealing with case constructions in an integrated way may be able to resolve the potential ambiguity using information from what other cases in the case frame are filled, while a uniform strategy would naturally tend to generate all the ambiguous alternatives, and this certainly was the case for FlexP. A specific example arises in the case of:

the messages Jones to Smith

Here, there are two relationships between Persons and Messages - sender and recipient. Since Smith is marked as the recipient, an integrated case strategy can tell that Jones must fill the other relationship, whereas FlexP because of its uniform strategy would get the Smith relation right, but flag an ambiguous relation for Jones.

Examples like these forced us to the conclusion that parsing case constructions efficiently and unambiguously in a flexible manner demands parsing techniques specific to case constructions. In turn, this caused us to review our design decision to use a uniform grammar based on linear patterns, which does not lend itself to such construction-specific parsing techniques. Since similar arguments

can be made against all uniform parsing methods, the idea of developing a parser based on a number of different parsing strategies suggested itself.

2.3. Experiments with a Multi-Strategy Approach to Parsing

Parsing using several different construction-specific strategies is a novel approach, so instead of trying to implement a full-scale parser immediately, we decided to try out the ideas in two simplified parsers, CASPAR and DYPAR. CASPAR was designed to show the suitability of construction specific techniques for ungrammatical input, while DYPAR served as a vehicle to investigate the control problems of coordinating several distinct parsing strategies. We describe both of them briefly below. Further details can be found in[5].

CASPAR was designed to use some of the insights about the flexible parsing of case constructions mentioned in the previous section. To keep things as simple as possible, CASPAR was designed only 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 course-registration at a university include:

cancel math 247
enroll Jim Campbell in English 224
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 preposition, or unmarked and identified by position, such as the position of direct object in the examples above.

In line with the construction-specific approach, 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 fillers 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.

While CASPAR was constructed in as simple a way as possible, the flexibility and robustness that

were 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 alone or in combination include:

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

  tStQtS enroll if you don't mind student 2476 in I think Economics 247.

- missing case markers:

  enroll Jim Campbell Economics 247.

- out of order cases:

  In Economics 247 Jim Campbell enroll.

- ambiguous cases:

  transfer Jim Campbell Economics 247 English 332.

Moreover, the construction-specific approach of CASPAR allowed it to deal with all these kinds of ambiguity without the inefficiencies and unnecessary ambiguities that arose with FlexP as described in the previous section.

While CASPAR concentrated on dealing with ungrammaticality through construction-specific strategies, our other experimental parser, DYPAR, concentrated on the control problems involved in combining several different parsing strategies. 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. These three strategies are:

- **A context-free semantic grammar component**, grouping domain information into hierarchical semantic categories useful in classifying individual words and phrases in the input language.

- **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.

These three strategies were combined into a single parser through the use of an applicative

condition-action cycle in which all matching rules were allowed to fire on each cycle. This allowed these three quite distinct types of strategy to work effectively together.

2.4. Non-Parsing Advantages of a Construction-Specific Approach

Besides showing the promise of a multi-strategy construction-specific approach to parsing for both grammatical and ungrammatical input, our experiments with CASPAR and DYPAR also showed the approach had other advantages, not directly involved in parsing. In particular, the approach made it straightforward to produce localized representations of unavoidable ambiguity, thus enhancing interaction with the user to resolve the ambiguity. In addition, the approach allows task-specific languages, defined in terms natural to the domain, to be used by the parser without a time-consuming compilation phase, thus significantly enhancing the language development process. The remainder of this section expands on these points. Further details can be found in[4]

If a flexible parser being used as part of an interactive system cannot correct ungrammatical input with reasonable certainty, then the system user must be involved in the resolution of the difficulty Experience with our first flexible parser, FlexP, suggested that the way requests for clarification in such situations are phrased makes a big difference in the ease and accuracy with which the user can correct his errors, and that the user is helped most by a request focusing as tightly as possible on the exact source and nature of the difficulty. As the following examples show, this type of focused interaction was very difficult to arrange with FlexP, but was straightforward using the construction-specific approach of CASPAR.

The following input is typical for the electronic mail system interface[1] with which FlexP was extensively used:

the messages from Fred Smith that arrived after Jon 5

The fact that this is not a complete sentence in FlexP's grammar causes no problem. The only real difficulty comes from "Jon", which should presumably be either "Jun" or "Jan". FlexP's spelling corrector can come to the same conclusion, so the output contains two complete parses which are passed onto the next stage of the mail system interface. The first of these parses looks like:

```
[DescriptionOf: Message
 Sender: [DescriptionOf: Person
          FirstName: fred
          Surname: smith
         ]
 AfterDate. [DescriptionOf: Date
             Month: january
             DayOfMonth: 5
            ]
]
```

This schematized property list style of representation should be interpreted in the obvious way.

If the next stage of the interface has access to other contextual information which allows it conclude that one or other of these parses was what was intended, then it can proceed to fulfill the user's request. Otherwise it has little choice but to ask a question involving paraphrases of each of the ambiguous interpretations, such as:

Do you mean:
1. the messages from Fred Smith that arrived after January 5
2. the messages from Fred Smith that arrived after June 5

Because it is not focused on the source of the error, this question gives the user very little help in seeing where the problem with his input actually lies.

One straightforward solution to the problem is to augment the output language with a special ambiguity representation. The output from our example might look like:

```
[DescriptionOf: Message
 Sender: [DescriptionOf: Person
          FirstName: fred
          Surname: smith
         ]
 AfterDate: [DescriptionOf: Date
             Month: [DescriptionOf: AmbiguitySet
                     Choices: (january june)
                    ]
             DayOfMonth: 5
            ]
]
```

This representation is exactly like the one above except that the Month slot is filled by an AmbiguitySet record. This record allows the ambiguity between january and june to be confined to the month slot where it belongs rather than expanding to an ambiguity of the entire input as in the first approach we discussed. By expressing the ambiguity set as a disjunction, it would be straightforward to generate from this representation a much more focused request for clarification such as:

Do you mean the messages from Fred Smith that arrived after **January** or **June** 5?

However, this approach only works if the ambiguity corresponds to an entire slot filler. Suppose, for example, that instead of mistyping the month, the user omitted or so completely garbled the preposition "from" that the parser effectively saw:

the messages Fred Smith that arrived after Jan 5

In the grammar used by FlexP for this particular application, the connection between Fred Smith and the message could have been expressed only by "from", "to", or "copied to" (or synonyms thereof). To represent the ambiguity, FlexP generates three complete parses isomorphic to the first output example above, except that Sender is replaced by Recipient and CC in the second and third parses respectively. Again, this form of representation does not allow the system to ask a focused question about the source of the ambiguity. The previous solution is not applicable because *the ambiguity lies in the structure of the parser output rather than at one of its terminal nodes*. Using a case notation, it is not permissible to put an "AmbiguitySet" in place of one of the deep case markers. To localize such ambiguities and avoid duplicate representation of unambiguous parts of the input, it is necessary to employ a representation like the one we designed for CASPAR.

```
[DescriptionOf: Message
 AmbiguousSlots: (
                  [PossibleSlots: (Sender Recipient CC)
                   SlotFiller: [DescriptionOf: Person
                                FirstName: fred
                                Surname: smith
                               ]
                  ]
                 )
 AfterDate: [DescriptionOf: Date
             Month: january
             DayOfMonth: 5
            ]
]
```

This example CASPAR output is similar to the two given previously, but instead of having a Sender slot, it has an AmbiguousSlots slot. The filler of this slot is a list of records, each of which specifies a SlotFiller and a list of PossibleSlots. The SlotFiller is a structure that would normally be the filler of a slot in the top-level description (of a message in this case), but the parser has been unable to determine exactly which higher-level slot it should fit into; the possibilities are given in PossibleSlots. With this representation, it is now straightforward to construct a directed question such as:

Do you mean the messages **from**, **to**, or **copied to** Fred Smith that arrived after January 5?

The adoption of such representations for ambiguity has profound implications for the parsing strategies employed by any parser that tries to produce them. For each type of construction that such a parser can encounter, the parser must "know" about all the structural ambiguities that the construction can give rise to, and must be prepared to detect and encode appropriately such

ambiguities when they arise. Construction-specific parsing techniques as used in CASPAR fit this requirement perfectly. Each construction-specific parsing strategy can encode detailed information about the types of structural ambiguity possible with that construction and incorporate the specific information necessary to detect and represent these ambiguities.

This section concludes with a brief note about language definition. As we described in the last annual report, FlexP had a language definition facility which allowed the designer of a task-specific language to define the language without having to know the exact details of FlexP's grammar formalism. This made it much easier to define such languages, but the facility turned out to be inconvenient to use in practice because of the time-consuming compilation phase necessary to transform the language definition in domain terms into FlexP's pattern-match rule formalism. This was particularly inconvenient when a large number of relatively minor changes need to be made, as is normal during language development.

For CASPAR, we implemented a similar language definition facility, but with one important difference - instead of compiling the language definitions into a different formalism, we designed CASPAR to interpret them directly. This made the language designer's job much easier, by letting him make the many small changes that are always necessary in the course of developing a language, without requiring him to go though a time-consuming compilation for each incremental change. The reason that it was possible to do this with CASPAR, and not with FlexP, relates directly to the construction-specific approach that CASPAR embodies - since the constructions CASPAR deals with correspond directly to those that are natural to the domain, direct interpretation of a language representation designed round these constructions was straightforward for CASPAR.

2.5. Further Applications of the Construction-Specific Approach

After reviewing the results of experiments with our two simple construction-specific multi-strategy parsers, CASPAR and DYPAR, it seemed to us that the construction-specific approach was promising enough to merit further investigation in the context of more powerful and realistic parsers. Accordingly, towards the end of the contract period which is the subject of this report, we embarked on the design of two new further parsers, which are intended to push the multi-strategy approach along two different dimensions.

The first of these new parsers, for which a preliminary design appears in[3] and[2], is intended to proceed further in the same direction as CASPAR and DYPAR, viz. in the direction of limited-domain natural language parsing by construction-specific techniques. The most important issue to be

tackled in this parser is the coordination of a number of diverse parsing techniques to produce an efficient, robust, and reliable unified parser.

The second parser we are planning is an attempt to apply these same construction-specific techniques to languages of the type commonly used in interactive interfaces. Important issues here involve the handling of positionally determined constituents, used much more widely in such languages than in natural languages (whether restricted-domain or more general), and the importance of efficiency - typical parsers for such languages, though not very flexible, are usually very fast, and to provide a reasonable alternative, our flexible parser must be able to compete along this dimension too. We expect work on both these parsers to continue through the remainder of this year.

3. Publications

The first of the publications listed below discusses the interface project within which FlexP was tested.

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, Stanford University, August, 1980, pp. 116-120.

2. Carbonell, J. G. and Hayes, P. J. Dynamic Strategy Selection in Flexible Parsing. Proc. of 19th Annual Meeting of the Assoc. for Comput. Ling., Stanford University, June, 1981.

3. Hayes, P. J. and Carbonell, J. G. Multi-Strategy Parsing and its Role in Robust Man-Machine Communication. Carnegie-Mellon University Computer Science Department, May, 1981.

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

5. Hayes, P. J. and Carbonell, J. G. Multi-Strategy Construction-Specific Parsing for Flexible Data Base Query and Update. Proc. Seventh Int. Jt. Conf. on Artificial Intelligence, Univ. of British Columbia, Vancouver, August, 1981, pp. 432-439.

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.

4. Professional Personnel

1. Philip J. Hayes
   D Sc, 1977, "Some Association-Based Techniques for Lexical Disambiguation by Machine".

2. George V. Mouradian

M Ph, 1978.

5. Interactions

Ongoing consultation with Dr. Jaime Carbonell, also a faculty member in the Computer Science Department of Carnegie-Mellon University, but not funded under this contract.