CMU-CS-84-133

Entity-Oriented Parsing

Philip J. Hayes

9 June 1984

Abstract

An entity-oriented approach to restricted-domain parsing is proposed. In this approach, the definitions of the structure and surface representation of domain entities are grouped together. Like semantic grammar, this allows easy exploitation of limited domain semantics. In addition, it facilitates fragmentary recognition and the use of multiple parsing strategies, and so is particularly useful for robust recognition of extragrammatical input. Several advantages from the point of view of language definition are also noted. Representative samples from an entity-oriented language definition are presented, along with a control structure for an entity-oriented parser, some parsing strategies that use the control structure, and worked examples of parses. A parser incorporating the control structure and the parsing strategies is currently under implementation.

To appear in Proceedings of Coling 84.

This research was sponsored by the Air Force Office of Scientific Research under Contract AFOSR-82-0219

1. Introduction

The task of typical natural language interface systems is much simpler than the general problem of natural language understanding. The simplifications arise because:

1. the systems operate within a highly restricted domain of discourse, so that a precise set of object types can be established, and many of the ambiguities that come up in more general natural language processing can be ignored or constrained away;

2. even within the restricted domain of discourse, a natural language interface system only needs to recognize a limited subset of all the things that could be said — the subset that its back-end can respond to.

The most commonly used technique to exploit these limited domain constraints is semantic grammar [1, 2, 9] in which semantically defined categories (such as <ship> or <ship-attribute>) are used in a grammar (usually ATN based) in place of syntactic categories (such as <noun> or <adjective>). While semantic grammar has been very successful in exploiting limited domain constraints to reduce ambiguities and eliminate spurious parses of grammatical input, it still suffers from the fragility in the face of extragrammatical input characteristic of parsing based on transition nets [4]. Also, the task of restricted-domain language definition is typically difficult in interfaces based on semantic grammar, in part because the grammar definition formalism is not well integrated with the method of defining the object and actions of the domain of discourse (though see [6]).

This paper proposes an alternative approach to restricted domain language recognition called *entity-oriented parsing*. Entity-oriented parsing uses the same notion of semantically-defined categories as semantic grammar, but does not embed these categories in a grammatical structure designed for syntactic recognition. Instead, a scheme more reminiscent of conceptual or case-frame parsers [3, 10, 11] is employed. An entity-oriented parser operates from a collection of definitions of the various entities (objects, events, commands, states, etc.) that a particular interface system needs to recognize. These definitions contain information about the internal structure of the entities, about the way the entities will be manifested in the natural language input, and about the correspondence between the internal structure and surface representation. This arrangement provides a good framework for exploiting the simplifications possible in restricted domain natural language recognition because:

1. the entities form a natural set of types through which to constrain the recognition semantically. The types also form a natural basis for the structural definitions of entities.

2. the set of things that the back-end can respond to corresponds to a subset of the domain entities (remember that entities can be events or commands as well as objects). So the goal of an entity-oriented system will normally be to recognize one of a "top-level" class of entities. This is analogous to the set of basic message patterns that the machine

translation system of Wilks [11] aimed to recognize in any input.

In addition to providing a good general basis for restricted domain natural language recognition, we claim that the entity-oriented approach also facilitates robustness in the face of extragrammatical input and ease of language definition for restricted domain languages. Entity-oriented parsing has the potential to provide better parsing robustness than more traditional semantic grammar techniques for two major reasons:

- The individual definition of all domain entities facilitates their independent recognition. Assuming there is appropriate indexing of entities through lexical items that might appear in a surface description of them, this recognition can be done bottom-up, thus making possible recognition of elliptical, fragmentary, or partially incomprehensible input. The same definitions can also be used in a more efficient top-down manner when the input conforms to the system's expectations.

- Recent work [5, 8] has suggested the usefulness of multiple construction-specific recognition strategies for restricted domain parsing, particularly for dealing with extragrammatical input. The individual entity definitions form an ideal framework around which to organize the multiple strategies. In particular, each definition can specify which strategies are applicable to recognizing it. Of course, this only provides a framework for robust recognition, the robustness achieved still depends on the quality of the actual recognition strategies used.

The advantages of entity-oriented parsing for language definition include:

- All information relating to an entity is grouped in one place, so that a language definer will be able to see more clearly whether a definition is complete and what would be the consequences of any addition or change to the definition.

- Since surface (syntactic) and structural information about an entity is grouped together, the surface information can refer to the structure in a clear and coherent way. In particular, this allows hierarchical surface information to use the natural hierarchy defined by the structural information, leading to greater consistency of coverage in the surface language.

- Since entity definitions are independent, the information necessary to drive recognition by the multiple construction-specific strategies mentioned above can be represented directly in the form most useful to each strategy, thus removing the need for any kind of "grammar compilation" step and allowing more rapid grammar development.

In the remainder of the paper, we make these arguments more concrete by looking at some fragments of an entity-oriented language definition, by outlining the control structure of a robust restricted-domain parser driven by such definitions, and by tracing through some worked examples of the parser in operation. These examples also shown describe some specific parsing strategies that exploit the control structures. A parser incorporating the control structure and the parsing strategies

is currently under implementation. Its design embodies our experience with a pilot entity-oriented parser that has already been implemented, but is not described here.

2. Example Entity Definitions

This section presents some example entity and language definitions suitable for use in entity-oriented parsing. The examples are drawn from the domain of an interface to a database of college courses. Here is the (partial) definition of a course.

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

For reasons of space, we cannot explain all the details of this language. In essence, a course is defined as a structured object with components: number, department, instructor, etc. (square brackets denote attribute/value lists, and round brackets ordinary lists). This definition is kept separate from the surface representation of a course which is defined to be a noun phrase with adjectives, postnominal cases, etc.. At a more detailed level, note the special purpose way of

specifying a course by its department juxtaposed with its number (e.g. Computer Science 101) is handled by an alternate pattern for the head of the noun phrase (dollar signs refer back to the components). This allows the user to say (redundantly) phrases like "CS 101 taught by Smith". Note also that the way the department of a course can appear in the surface representation of a course is specified in terms of the CourseDepartment component (and hence in terms of its type, CollegeDepartment) rather than directly as an explicit surface representation. This ensures consistency throughout the language in what will be recognized as a description of a department. Coupled with the ability to use general syntactic descriptors (like NounPhrase in the description of a SurfaceRepresentation), this can prevent the kind of patchy coverage prevalent with standard semantic grammar language definitions.

Subsidiary objects like CollegeDepartment are defined in similar fashion.

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

CollegeCourse will also be involved in higher-level entities of our restricted domain such as a command to the data base system to enrol a student in a course.

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

These examples also show how all information about an entity, concerning both fundamental structure and surface representation, is grouped together and integrated. This supports the claim that entity-oriented language definition makes it easier to determine whether a language definition is complete.

3. Control Structure for a Robust Entity-Oriented Parser

The potential advantages of an entity-oriented approach from the point of view of robustness in the face of ungrammatical input were outlined in the introduction. To exploit this potential while maintaining efficiency in parsing grammatical input, special attention must be paid to the control structure of the parser used. Desirable characteristics for the control structure of any parser capable of handling ungrammatical as well as grammatical input include:

- the control structure allows grammatical input to be parsed straightforwardly without considering any of the possible grammatical deviations that could occur;
- the control structure enables progressively higher degrees of grammatical deviation to be considered when the input does not satisfy grammatical expectations;
- the control structure allows simpler deviations to be considered before more complex deviations.

The first two points are self-evident, but the third may require some explanation. The problem it addresses arises particularly when there are several alternative parses under consideration. In such cases, it is important to prevent the parser from considering drastic deviations in one branch of the parse before considering simple ones in the other. For instance, the parser should not start hypothesizing missing words in one branch when a simple spelling correction in another branch

would allow the parse to go through.

We have designed a parser control structure for use in entity-oriented parsing which has all of the characteristics listed above. This control structure operates through an agenda mechanism. Each item of the agenda represents a different *continuation* of the parse, i.e. a partial parse plus a specification of what to do next to continue that partial parse. With each continuation is associated an integer *flexibility level* that represents the degree of grammatical deviation implied by the continuation. That is, the flexibility level represents the degree of grammatical deviation in the input if the continuation were to produce a complete parse without finding any more deviation. Continuations with a lower flexibility are run before continuations with a higher flexibility level. Once a complete parse has been obtained, continuations with a flexibility level higher than that of the continuation which resulted in the parse are abandoned. This means that the agenda mechanism never activates any continuations with a flexibility level higher than the level representing the lowest level of grammatical deviation necessary to account for the input. Thus effort is not wasted exploring more exotic grammatical deviations when the input can be accounted for by simpler ones. This shows that the parser has the first two of the characteristics listed above.

In addition to taking care of alternatives at different flexibility levels, this control structure also handles the more usual kind of alternatives faced by parsers — those representing alternative parses due to local ambiguity in the input. Whenever such an ambiguity arises, the control structure duplicates the relevant continuation as many times as there are ambiguous alternatives, giving each of the duplicated continuations the same flexibility level. From there on, the same agenda mechanism used for the various flexibility levels will keep each of the ambiguous alternatives separate and ensure that all are investigated (as long as their flexibility level is not too high). Integrating the treatment of the normal kind of ambiguities with the treatment of alternative ways of handling grammatical deviations ensures that the level of grammatical deviation under consideration can be kept the same in locally ambiguous branches of a parse. This fulfills the third characteristic listed above.

Flexibility levels are additive, i.e. if some grammatical deviation has already been found in the input, then finding a new one will raise the flexibility level of the continuation concerned to the sum of the flexibility levels involved. This ensures a relatively high flexibility level and thus a relatively low likelihood of activation for continuations in which combinations of deviations are being postulated to account for the input.

Since space is limited, we cannot go into the implementation of this control structure. However, it is possible to give a brief description of the control structure primitives used in programming the parser.

Recall first that the kind of entity-oriented parser we have been discussing consists of a collection of recognition strategies. The more specific strategies exploit the idiosyncratic features of the entities/construction types they are specific to, while the more general strategies apply to wider classes of entities and depend on more universal characteristics. In either case, the strategies are pieces of (Lisp) program rather than more abstract rules or networks. Integration of such strategies with the general scheme of flexibility levels described above is made straightforward through a special *split* function which the control structure supports as a primitive. This split function allows the programmer of a strategy to specify one or more alternative continuations from any point in the strategy and to associate a different flexibility increment with each of them. The implementation of this statement takes care of restarting each of the alternative continuations at the appropriate time and with the appropriate local context.

Some examples should make this account of the control structure much clearer. The examples will also present some specific parsing strategies and show how they use the split function described above. These strategies are designed to effect robust recognition of extragrammatical input and efficient recognition of grammatical input by exploiting entity-oriented language definitions like those in the previous section.

4. Example Parses

Let us examine first how a simple data base command like:

Enrol Susan Smith in CS 101

might be parsed with the control structure and language definitions presented in the two previous sections. We start off with the top-level parsing strategy, RecognizeAnyEntity. This strategy first tries to identify a top-level domain entity (in this case a data base command) that might account for the entire input. It does this in a bottom-up manner by indexing from words in the input to those entities that they could appear in. In this case, the best indexer is the first word, 'enrol', which indexes EnrolCommand. In general, however, the best indexer need not be the first word of the input and we need to consider all words, thus raising the potential of indexing more than one entity. In our example, we would also index CollegeStudent, CollegeCourse, and CollegeDepartment. However, these are not top-level domain entities and are subsumed by EnrolCommand, and so can be ignored in favour of it.

Once EnrolCommand has been identified as an entity that might account for the input, RecognizeAnyEntity initiates an attempt to recognize it. Since EnrolCommand is listed as an imperative case frame, this task is handled by the ImperativeCaseFrame recognizer strategy. In contrast to the bottom-up approach of RecognizeAnyEntity, this strategy tackles its more specific task

in a top-down manner using the case frame recognition algorithm developed for the CASPAR parser [8]. In particular, the strategy will match the case frame header and the preposition 'in', and initiate recognitions of fillers of its direct object case and its case marked by 'in'. These subgoals are to recognize a CollegeStudent to fill the Enrollee case on the input segment "Susan Smith'" and a CollegeCourse to fill the EnrolIn case on the segment "CS 101". Both of these recognitions will be successful, hence causing the ImperativeCaseFrame recognizer to succeed and hence the entire recognition. The resulting parse would be:

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

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

The last example was completely grammatical and so did not require any flexibility. After an initial bottom-up step to find a dominant entity, that entity was recognized in a highly efficient top-down manner. For an example involving input that is ungrammatical (as far as the parser is concerned), consider:

Place Susan Smith in computer science for freshmen

There are two problems here: we assume that the user intended 'place' as a synonym for 'enrol', but that it happens not to be in the system's vocabulary; the user has also shortened the grammatically acceptable phrase, 'the computer science course for freshmen', to an equivalent phrase not covered by the surface representation for CollegeCourse as defined earlier. Since 'place' is not a synonym for 'enrol' in the language as presently defined, the RecognizeAnyEntity strategy cannot index EnrolCommand from it and hence cannot (as it did in the previous example) initiate a top-down recognition of the entire input.

To deal with such eventualities, RecognizeAnyEntity executes a split statement specifying two continuations immediately after it has found all the entities indexed by the input. The first continuation has a zero flexibility level increment. It looks at the indexed entities to see if one subsumes all the others. If it finds one, it attempts a top-down recognition as described in the previous example. If it cannot find one, or if it does and the top-down recognition fails, then the continuation itself fails. The second continuation has a positive flexibility increment and follows a more robust bottom-up approach described below. This second continuation was established in the

previous example too, but was never activated since a complete parse was found at the zero flexibility level. So we did not mention it. In the present example, the first continuation fails since there is no subsuming entity, and so the second continuation gets a chance to run. Instead of insisting on identifying a single top-level entity, this second continuation attempts to recognize all of the entities that are indexed in the hope of later being able to piece together the various fragmentary recognitions that result. The entities directly indexed are CollegeStudent by "Susan" and "Smith",[1] CollegeDepartment by "computer" and "science", and CollegeClass by "freshmen". So a top-down attempt is made to recognize each of these entities. We can assume these goals are fulfilled by simple top-down strategies, appropriate to the SurfaceRepresentation of the corresponding entities, and operating with no flexibility level increment.

Having recognized the low-level fragments, the second continuation of RecognizeAnyEntity now attempts to unify them into larger fragments, with the ultimate goal of unifying them into a description of a single entity that spans the whole input. To do this, it takes adjacent fragments pairwise and looks for entities of which they are both components, and then tries to recognize the subsuming entity in the spanning segment. The two pairs here are CollegeStudent and CollegeDepartment (subsumed by CollegeStudent) and CollegeDepartment and CollegeClass (subsumed by CollegeCourse).

To investigate the second of these pairings, RecognizeAnyEntity would try to recognize a CollegeCourse in the spanning segment 'computer science for freshmen' using an elevated level of flexibility. This goal would be handled, just like all recognitions of CollegeCourse, by the NominalCaseFrame recognizer. With no flexibility increment, this strategy fails because the head noun is missing. However, with another flexibility increment, the recognition can go through with the CollegeDepartment being treated as an adjective and the CollegeClass being treated as a postnominal case — it has the right case marker, "for", and the adjective and post-nominal are in the right order. This successful fragment unification leaves two fragments to unify — the old CollegeStudent and the newly derived CollegeCourse.

There are several ways of unifying a CollegeStudent and a CollegeCourse — either could subsume the other, or they could form the parameters to one of three database modification commands: EnrolCommand, WithdrawCommand, and TransferCommand (with the obvious interpretations). Since the commands are higher level entities than CollegeStudent and CollegeCourse, they would be preferred as top-level fragment unifiers. We can also rule out TransferCommand in favour of the first two because it requires two courses and we only have one. In addition, a recognition of

[1] We assume we have a complete listing of students and so can index from their names.

EnrolCommand would succeed at a lower flexibility increment than WithdrawCommand,[2] since the preposition 'in' that marks the CollegeCourse in the input is the correct marker of the EnrolIn case of EnrolCommand, but is not the appropriate marker for WithdrawFrom, the course-containing case of WithdrawCommand. Thus a fragment unification based on EnrolCommand would be preferred. Also, the alternate path of fragment amalgamation — combining CollegeStudent and CollegeDepartment into CollegeStudent and then combining CollegeStudent and CollegeCourse — that we left pending above cannot lead to a complete instantiation of a top-level database command. So RecognizeAnyEntity will be in a position to assume that the user really intended the EnrolCommand.

Since this recognition involved several significant assumptions, we would need to use focused interaction techniques [7] to present the interpretation to the user for approval before acting on it. Note that if the user does approve it, it should be possible (with further approval) to add 'place' to the vocabulary as a synonym for 'enrol' since 'place' was an unrecognized word in the surface position where 'enrol' should have been.

For a final example, let us examine an extragrammatical input that involves continuations at several different flexibility levels:

Transfer Smith from Compter Science 101 Economics 203

The problems here are that 'Computer' has been misspelt and the preposition 'to' is missing from before 'Economics'. The example is similar to the first one in that RecognizeAnyEntity is able to identify a top-level entity to be recognized top-down, in this case, TransferCommand. Like EnrolCommand, TransferCommand is an imperative case frame, and so the task of recognizing it is handled by the ImperativeCaseFrame strategy. This strategy can find the preposition 'from', and so can initiate the appropriate recognitions for fillers of the OutOfCourse and Student cases. The recognition for the student case succeeds without trouble, but the recognition for the OutOfCourse case requires a spelling correction.

Whenever a top-down parsing strategy fails to verify that an input word is in a specific lexical class, there is the possibility that the word that failed is a misspelling of a word that would have succeeded. In such cases, the lexical lookup mechanism executes a split statement.[3] A zero increment branch

---

[2] This relatively fine distinction between EnrolCommand and WithdrawCommand, based on the appropriateness of the preposition 'in', is problematical in that it assumes that the flexibility level would be incremented in very fine-grained steps. If that was impractical, the final outcome of the parse would be ambiguous between an EnrolCommand and a WithdrawCommand and the user would have to be asked to make the discrimination.

[3] If this causes too many splits, an alternative is only to do the split when the input word in question is not in the system's lexicon at all.

fails immediately, but a second branch with a small positive increment tries spelling correction against the words in the predicted lexical class. If the correction fails, this second branch fails, but if the correction succeeds, the branch succeeds also. In our example, the continuation involving the second branch of the lexical lookup is highest on the agenda after the primary branch has failed. In particular, it is higher than the second branch of RecognizeAnyEntity described in the previous example, since the flexibility level increment for spelling correction is small. This means that the lexical lookup is continued with a spelling correction, thus resolving the problem. Note also that since the spelling correction is only attempted within the context of recognizing a CollegeCourse — the filler of OutOfCourse — the target words are limited to course names. This means spelling correction is much more accurate and efficient than if correction were attempted against the whole dictionary.

After the OutOfCourse and Student cases have been successfully filled, the ImperativeCaseFrame strategy can do no more without a flexibility level increment. But it has not filled all the required cases of TransferCommand, and it has not used up all the input it was given, so it splits and fails at the zero-level flexibility increment. However, in a continuation with a positive flexibility level increment, it is able to attempt recognition of cases without their marking prepositions. Assuming the sum of this increment and the spelling correction increment are still less than the increment associated with the second branch of RecognizeAnyEntity, this continuation would be the next one run. In this continuation, the ImperativeCaseFrameRecognizer attempts to match unparsed segments of the input against unfilled cases. There is only one of each, and the resulting attempt to recognize 'Economics 203' as the filler of IntoCourse succeeds straightforwardly. Now all required cases are filled and all input is accounted for, so the ImperativeCaseFrame strategy and hence the whole parse succeeds with the correct result.

For the example just presented, obtaining the ideal behaviour depends on careful choice of the flexibility level increments. There is a danger here that the performance of the parser as a whole will be dependent on iterative tuning of these increments, and may become unstable with even small changes in the increments. It is too early yet to say how easy it will be to manage this problem, but we plan to pay close attention to it as the parser comes into operation.

5. Conclusion

Entity-oriented parsing has several advantages as a basis for language recognition in restricted domain natural language interfaces. Like techniques based on semantic grammar, it exploits limited domain semantics through a series of domain-specific entity types. However, because of its suitability for fragmentary recognition and its ability to accommodate multiple construction-specific parsing

strategies, it has the potential for greater robustness in the face of extragrammatical input than the usual semantic grammar techniques. In this way, it more closely resembles conceptual or case-frame parsing techniques. Moreover, entity-oriented parsing offers advantages for language definition because of the integration of structural and surface representation information and the ability to represent surface information in the form most convenient to drive construction-specific recognition strategies directly.

A pilot implementation of an entity-oriented parser has been completed and provides preliminary support for our claims. However, a more rigorous test of the entity-oriented approach must wait for the more complete implementation currently being undertaken. The agenda-style control structure we plan to use in this implementation is described above, along with some parsing strategies it will employ and some worked examples of the strategies and control structure in action.

Acknowledgements

The ideas in this paper benefited considerably from discussions with other members of the Multipar group at Carnegie-Mellon Computer Science Department, particularly Jaime Carbonell, Jill Fain, and Steve Minton. Steve Minton was a co-designer of the control structure presented above, and also found an efficient way to implement the split function described in connection with that control structure.

References

1. 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.

2. Burton, R. R. Semantic Grammar: An Engineering Technique for Constructing Natural Language Understanding Systems. BBN Report 3453, Bolt, Beranek, and Newman, Inc., Cambridge, Mass., December, 1976.

3. Carbonell, J. G., Boggs, W. M., Mauldin, M. L., and Anick, P. G. The XCALIBUR Project: A Natural Language Interface to Expert Systems. Proc. Eighth Int. Jt. Conf. on Artificial Intelligence, Karlsruhe, August, 1983.

4. Carbonell, J. G. and Hayes, P. J. "Recovery Strategies for Parsing Extragrammatical Language." *Computational Linguistics 10* (1984).

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

6. Grosz, B. J. TEAM: A Transportable Natural Language Interface System. Proc. Conf. on Applied Natural Language Processing, Santa Monica, February, 1983.

7. Hayes P. J. A Construction Specific Approach to Focused Interaction in Flexible Parsing. Proc. of 19th Annual Meeting of the Assoc. for Comput. Ling., Stanford University, June, 1981, pp. 149-152.

8. 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.

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

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

11. Wilks, Y. A. Preference Semantics. In *Formal Semantics of Natural Language*, Keenan, Ed.,Cambridge University Press, 1975.