HECTOR: A Logic Based Parser, Semantic Interpreter, and Planner

by

Kenneth L. Ingham
William S. Ziegler
George F. Luger
Department of Computer Science
University of New Mexico
Albuquerque, New Mexico
USEnet: ...ucbvax!unmvax!hector

March 1983

Abstract

In this paper, we describe our project to design a program that uses natural language to communicate with a general planner. We have adopted predicate logic as a representational formalism able to support: 1) English language syntactical analysis, 2) the creation of the semantic meanings necessary to disambiguate complex situations, 3) the formation of plans. "Hector" is learning to manipulate objects in and answer questions about his environment (a house). A project such as Hector has applications wherever a general planner can be used.

Acknowledgements

We would like to thank the Space Technology Branch of NASA Ames Research Center for a co-op position held by Kenneth Ingham, when the initial work on the parser was done. We would also like to thank David Warren for his help in implementing WARPLAN. A special thanks is owed to James P. Hogan, the real father of Hector.

Primary Topic: Problem Solving & Inference.

1. Who Is Hector?

In "The Two Faces of Tomorrow"(4), James P. Hogan describes an artificial intelligence project called "Hector", where researchers are trying to teach a computer the basics of preparing a breakfast. They have a graphics display of a kitchen, and in it Hector attempts to prepare fried eggs. Because of complications described in the book, they were not able to complete Hector. We feel that, based on the formalism of predicate logic, we can create Hector.

A primary goal of our research has been to examine the feasibility of adopting one representational formalism and designing a problem solving system around it. We have selected the predicate calculus(6) as the representation tool and Prolog(10) for the control regime. The goal of our project has been to carry out all aspects of the system in this single formalism. There is now research (references below) that use logic for various parts of this endeavor and the time is right for unifying this research. There are three parts to our project: 1) the syntactic parser, described in Section 3.1, 2) the semantic interpreter, Section 3.2, and 3) the planner, Section 3.3.

Winograd (14) has provided a valuable source of problems for Hector to solve, and indeed, we have used SHRDLU as a benchmark for our own successes. Winograd places SHRDLU in a blocks world where it is able to manipulate blocks in the simple environment of a table. Hector is in a more complex environment, his house, and can manipulate such things as tables and chairs. We have based our own efforts on getting Hector to perform tasks similar to those given to SHRDLU. At present, he can handle most of the tasks described in Winograd(14).

2. What Can Hector Do?

Below is a sample run of Hector using CProlog(10). The prompt from Hector is "What now?". The sentences that follow the prompt tell Hector about his environment, "The table is in the dining room.", ask him about the environment, "Where are you?", or finally, ask him to perform a task, "Go to the hall."

```
CProlog version 1.1
| ?- start.
Hector here, here is my current environment:
I am in the kitchen.
I am on the kitchen floor.
That's it.
What now? The table is in the dining room.
Ok.
What now? The yellow stool is in the hall.
Ok.
What now? Is the kitchen light on?
No.
What now? Turn it on.
I went to the kitchen light switch.
I turned on the light.
What now? The red chair is in the kitchen.
Ok.
What now? Put it beside the table.
I went to the red chair.
I pushed the red chair to the kitchen door.
I went to the kitchen door.
I moved the red chair from the kitchen to the dining room.
I went from the kitchen to the dining room.
I pushed the red chair to the table.
```

```
What now? Go to the table.
I went to the table.
What now? Where are you?
I am in the dining room.
I am on the dining room floor.
I am beside the table.
What now? Is the yellow stool in the living room?
No.
The yellow stool is in the hall.
What now? Climb on it.
I went to the dining room door.
I went from the dining room to the living room.
I went to the hallway door.
I went from the living room to the hall.
I went to the yellow stool.
I climbed on the yellow stool.
What now? Put it in the living room.
I climbed off the yellow stool.
I went to the yellow stool.
I pushed the yellow stool to the hallway door.
I went to the hallway door.
I moved the yellow stool from the hall to the living room.
What now? Where is the red chair?
The red chair is in the dining room.
The red chair is beside the table.
What now? What is in the dining room?
The table is in the dining room.
The red chair is in the dining room.
What now? Is the kitchen light on?
Yes.
What now? Go to the table.
I went from the hall to the living room.
I went to the dining room door.
I went from the living room to the dining room.
I went to the table.
What now? Turn off the kitchen light.
I went to the kitchen door.
I went from the dining room to the kitchen.
I went to the kitchen light switch.
I turned off the kitchen light.
What now? Bye.
[ Prolog execution halted ]
```

3. How Does He Do It?

When a sentence is given to Hector, several things happen. First, it is parsed to check syntax and put into a structure that is easier to handle than English. Next, semantic rules are applied to the structure, and a decision is made on what needs to be done. At this point, if a task is to be performed, the general planner is called. If it is a request for information, or a statement about his environment, it is handled directly. The next three subsections describe in more detail these aspects of our research problem.

3.1 Hector's Linguistic Abilities.

In recent publications Colmerauer(1), Dahl(2), Kowalski(6), McCord(8), Periera and Warren(9) have laid down a theoretical framework for parsing natural language statements using the formalism of logic, especially the subset of predicate logic called Horn clause logic(6).

The parser that we use is based on this framework and described by Ingham(5). We see generation and parsing of sentences as two special cases of one set of general logic rules describing English syntax. This view has two important advantages: 1) Since the same code both generates and parses sentences, we are able to reduce by about one half the amount of code needed; 2) we preserve logic as the sole representational formalism throughout all of Hector.

The general syntax rules are entered into the computer as assertions on how to build sentences. This idea is not new, rules similar to those we use will be found in any grammar textbook. The ones we use are based on Liles(7), who put his rules into a form similar to first order predicate logic. Since we are doing our work in predicate logic, there were few problems getting the computer to accept them.

The parser uses a concept similar to the definite clause grammar (DCG) described by Periera & Warren(9). This is the sole procedural mechanism for parsing, using logical inference in a restricted form of resolution. Besides the elegance and economy the DCG gives to our English language analysis code, it is also a form of logic, which makes it a consistent part of the entire Hector system.

The syntax rules contain a template for a parse tree. By supplying this framework, the parser, using a form of resolution, is able to immediately eliminate choices that do not pertain. In only looking at the parts that are relevant, we are able to keep the parser efficient.

Below is a small example of what is in the parser to show how resolution is used to our advantage. The adjective phrases show how list length and structure formation are both somewhat dependent of words supplied, but also follow a common format. This dependency on the input sentence gives the resolution theorem prover enough to make a quick match, while still retaining the capability to represent a large number of sentences.

```
adjective_phrase([Adj],ap(adj(Adj))) :- adjective(Adj,_).
adjective_phrase([I,A],ap(intens(I) plus adj(A))) :-
        intensifier(I),
        adjective(A,_).
```

3.2 The Heart of Hector.

The tree produced by the parser is similar to the one described by Liles(7). However, it is not in a form that the planner can easily use. In the heart of Hector, the parse tree is pulled apart, semantic rules are applied, and a structure is built. This structure can either be executed directly, as in a question or statement, or passed on to the planner.

In this section, we also handle all of Hector's replies. When the planner produces a plan, besides being executed, it is printed out as statements describing what Hector did. This is done by translating the plan into a structure that is then passed on to the parser for generation of sentences.

As was noted in Section 2, there are three types of sentences that are given to Hector. Declarative sentences are used to tell Hector about his world. They are converted into a form that the planner can use, and asserted into the database. Negation in a declarative sentence causes the fact to be removed.

Secondly, imperative sentences are requests for Hector to perform some action. These sentences are converted into a request for the planner. This conversion requires use of some general semantic rules to determine what we want Hector to do. Then the planner is called to generate the sequence of actions necessary to perform the task.

Finally, questions ask him about his world, and are normally answered directly. Questions about his world are converted into a form similar to that of declarative sentences. Then an attempt is made to match the question with a fact in the database. The response given depends on the success of that match. If the response is negative, a search is made for appropriate additional information. In the example, when Hector was asked "Is the yellow stool in the living room?", he answers, "No. The yellow stool is in the hall."

Also in this section, pronoun references are handled. The pronoun resolution strategy that we use is not complete, but will work for all of the cases we have encountered so far. Everything that is done, either by Hector or the person talking to Hector, is stored as an event. These events contain the parse structure and whether it was done by Hector or the person talking to him. Events also have a link to whatever event they cause. The pronoun resolution is handled by looking back through past events, and trying to match the properties of the pronoun to the properties of one of the nouns. Sentences are scanned in the order: subject, indirect object, direct object.

The properties that nouns and pronouns have are stored in a dictionary. With each noun, we have: concrete/abstract, animate/inanimate, human/nonhuman. Also, we know from the parsing whether the noun is singular or plural. For the matching, the pronoun must have the same properties as the noun. For example, "he" would match with "John" because they are both human and singular. "It" would not, however, match with "Mary", because "it" is inanimate, and "Mary" is a human.

3.3 Hector's Planning Abilities.

The planning portion of our program is based on a research formalism proposed by Warren, called WARPLAN(13) and on ideas of Kowalski(6). The sets of logic statements produced by the above section make up a description of the world; the task or problem to be solved will also be stated as sets of logic clauses. A particular problem is then described by giving an initial state and a desired goal state. The problem solver must then generate a plan for a sequence of actions that transform the world from the initial state to the goal state.

The example in Section 2 uses our modified form of WARPLAN to generate a solution plan for Hector. We have modified ideas from other similar planners for use in our database (STRIPS(3), ABSTRIPS(11), and LAWALY(12)). The planning produces results that are logic statements. As remarked above, these logic statements are passed back to the heart of Hector and produce the English language trace that describes the steps Hector is taking. The data base is modified as the planner generates the solution to the given problem.

4. Conclusions

Since we have been working on Hector, his English and general knowledge of the world have expanded rapidly. As a benchmark, we expect him to reach SHRDLU's level soon. We believe that our method of using logic shows promise for future applications, wherever a general planner is to be used in conjunction with natural language.

We have implemented Hector in Prolog. We chose Prolog because it is an efficient method of doing our programming in logic. We find programming in Prolog straightforward, clear, and easy to modify. Since Prolog is based on a resolution theorem prover, we were able to use the resolution to our advantage in keeping the system efficient.

5. References

(1) Colmerauer, A., Metamorphosis Grammars, in L. Bolc (ed), *Natural Language Communication with Computers*, New York: Springer, 1978.

(2) Dahl, V., Quantification in a Three Valued Logic for Natural Language Question Answering Systems, in *Proceedings IJCAI-79*, 1979.

(3) Fikes, R.E., and Nilsson, N.J., STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving, *Artificial Intelligence*, 1971.

(4) Hogan, James P., *The Two Faces of Tomorrow*, Del Rey, 1979.

(5) Ingham, Kenneth L., *The Logical Way to Parse English*, submission to IJCAI-83.

(6) Kowalski, R., *Logic for Problem Solving*, Amsterdam: North Holland, 1979.

(7) Liles, B. L., *An Introductory Transformational Grammar*, Prentice Hall, 1971.

(8) McCord, M.C., Using Slots and Modifiers in Logic Grammars for Natural Language, *Artificial Intelligence*, 18,327-367, 1982.

(9) Periera, F.C.N., and Warren, D.H.D., Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks, *Artificial Intelligence*, 13, 1980.

(10) Periera, F.C.N., *CProlog User's Manual, Version 1.1*, EdCAAD, Dept of Architecture, University of Edinburgh, 1982.

(11) Sacerdoti, E.D., Planning in a Hierarchy of Abstraction Spaces, *Artificial Intelligence*, 1974.

(12) Siklossy, L. and Dreussi, J., An Efficient Robot Planner which Generates its own Procedures, in *Proceedings of IJCAI-73*, 1973.

(13) Warren, D.H.D., *WARPLAN: A System for Generating Plans*, Univ. of Edinburgh Artificial Intelligence Dept., Tech. Report, 1977.

(14) Winograd, T., *Procedures as a Representation for Data in a Computer Program for Understanding Natural Language*, MIT Ph.D. thesis in Mathematics, Feb. 1971.