Director -- An Interpreter for Rule-based Programs

CUCS-231-86

Galina Datskovsky Moerdler
Department of Computer Science
Columbia University
New York, NY 10027

J. Robert Ensor
4F-607
AT&T Bell Laboratories
Holmdel, NJ 07733

ABSTRACT

Programs interacting with users via natural language interfaces generally require more sophisticated control structures than those needed by programs interacting with users through less flexible mechanisms. This paper describes our development of Director, an interpreter (inference engine) for rule-based programs. Providing an efficient combination of forward and backward chaining, heuristic and user control of inference, and ready access to portions of its internal structure, Director facilitates the construction of systems with natural language interfaces as well as other rule-based systems in which queries are expensive.

This research was partially supported by Office of Naval Research grant N00014-82-K-0256.

Director – An Interpreter for Rule-based Programs

ABSTRACT

Programs interacting with users via natural language interfaces generally require more sophisticated control structures than those needed by programs interacting with users through less flexible mechanisms. This paper describes our development of Director, an interpreter (inference engine) for rule-based programs. Providing an efficient combination of forward and backward chaining, heuristic and user control of inferences, and ready access to portions of its internal structure, Director facilitates the construction of systems with natural language interfaces as well as other rule-based systems in which queries are expensive.

1. INTRODUCTION

We have been investigating natural language interfaces for rule-based expert systems. During this study we have seen that programs interacting with users via natural language interfaces generally require more sophisticated control structures than those needed by programs interacting with users through less flexible mechanisms. This paper describes our development of Director, an interpreter (inference engine) for rule-based programs, which provides control structures facilitating the construction of natural language interfaces.

A typical rule-based expert system is constructed as a knowledge base and an associated interpreter (inference engine) [Ha 85], as indicated in Figure 1. The knowledge base is a collection of facts and production rules. A fact is a (name value) pair indicating the value of the object name; and a production rule is a statement of the form

IF premise THEN action.

Such a rule stores the knowledge that if the premise (left-hand-side) is true then the action (right-hand-side) should be performed. The interpreter controls the execution of the expert system by selecting and evaluating rules. As these rules are evaluated, they alter the values of the facts and generate input and output.

Expert systems commonly receive user input through menus, which are rigid, often stilted interfaces. When information is needed, the system typically either requests that the user supply a specified value or poses a question, with a selection of answers from which the user chooses the one that he/she thinks best corresponds to the correct answer [Da 84]. In such systems, information is solicited as it is needed, with little regard for the human user. Unfortunately, these interfaces are often tedious or awkward and may even limit the capabilities of associated expert systems [Po 82].

Natural language interfaces for expert systems are being designed to solve some of the problems posed by menus, thus providing the user greater flexibility and control during consultation sessions with the expert system [Da 85]. This flexibility, however, may cause some difficulties in constructing the expert system itself. A program with a natural language interface may be forced to accept facts in more or less unconstrained order, as the user volunteers information at arbitrary times. This volunteering of information creates conflicts between two goals we have for our expert systems. The first goal is that the system be responsive to the user input; and the second goal is that the system maintain reasonable continuity and focus in interactions despite the unconstrained order of user input. That is, we want the system to be responsive, but coherent. Additionally, we feel that the system should ask only necessary questions to arrive at its conclusions. Redundant or irrelevant questions detract from the system's natural language capabilities.

A rule-based program executes via the evaluation of its rules. These evaluations are controlled by the program's interpreter, which chooses which rules to evaluate according to some strategy. The system queries to the user are generated as the rules attempt to determine the values of various data. In these programs, therefore, our behavior goals can be realized through suitable control of rule firings, i.e., through an appropriate interpreter. Common interpreters for rule-based systems are based on sequential statement evaluation (e.g., [Bo 83]), forward chaining (e.g., [Fo 81]), or backward chaining (e.g., [va 81]). Sequential control, often used as the basis for specialized user-programmed control structures, is of little direct assistance in building rule-based systems. Systems that are restricted to forward chaining inference are difficult to make responsive to user queries of the form "Could so and so be true?" This is because forward chaining systems focus on deriving inferences from a set of facts, rather than investigating hypotheses. Systems restricted to backward chaining often do not allow a user to volunteer information, ignoring inferences from the new information.

Using a combination of forward and backward chaining, an interpreter can provide the basis of a flexible control mechanism for building expert systems that carefully control their input/output activities. However, intelligent control of a system's input/output requires more than the availability of both forward and backward chaining in its interpreter. The system also needs to record information describing its input, and then use this information in the control of the inference process. For example, an inference process might base its rule selection on the order a set of facts were entered. Finally, if intelligent control of input/output (such as that in a natural language interface) is to be built outside the interpreter, then the interpreter must provide appropriate information about itself to this high-level control mechanism. For example, a natural language interface might require knowledge of various relationships among the facts and rules available to the interpreter in order to answer some user queries. We have developed Director to provide an efficient combination of forward and backward chaining, the use of input attributes in controlling inferences, and ready access to portions of its internal structure for use by natural language interfaces.

More generally, Director supports the construction of any rule-based system in which queries are expensive. For example, a system that serves as a consultation aid for professionals (e.g., an aid for tax advisers) should not pose redundant or superfluous questions. Some expert systems for medical diagnosis also need to limit their queries. Often information about a patient can be obtained only through bothersome or even risky methods. Performing an unnecessary biopsy, even if it is a routine procedure, may be an expensive mistake and may expose the patient to the possibility of complications. Similarly, an expert system whose problem domain involves data from a hostile environment often has difficulty obtaining these data. For example, a system for diagnosing faults in an undersea cable becomes too expensive if it requests unnecessary probes.

The remainder of this paper has the following organization: Section 2 outlines the structure of a rule-based system that uses Director. The implementation of Director is described in Section 3. Section 4 describes techniques used to control the inferencing of Director, and Section 5 presents the mechanisms used to display the rules used. Finally, Section 6 presents the status of our implementation and experimental work, with some concluding remarks.

2. SYSTEM STRUCTURE

In the typical rule-based expert system, program execution proceeds as the interpreter evaluates the system's rules. That is, the interpreter acts as the program controller by executing a simple loop: select a rule for evaluation and evaluate it. Since each rule is selected according to the selection procedure contained within the interpreter, this procedure influences the structure of the rules and the control information that must be explicitly encoded into the system. Indeed there is probably no major expert system in which the rules are independent of their interpreter [Du 84]. Hence, our description of Director includes a brief description of the rules that it evaluates.

2.1 Rules

Director is designed to interpret rules that have been defined by a system called Describe [Bl 84]. Each rule is invoked as a function, whose body is an if-then form in which the premise and the action may be arbitrary Lisp s-expressions. Although the premise and the action of a rule may have arbitrary side effects, we assume here that the rules have a more restricted form.

Each rule premise is restricted to data base queries, i.e., the examination of the values of facts. The value of a fact may be added to the data base in only two ways: either through the action of a rule or through user input. Any fact that is not added by the action of a rule has an associated query procedure so that the user can supply its values. This query procedure is invoked if the premise of a rule tries to examine the fact's value, and the value is not present in the data base. Director automatically maintains the mappings between the rules and the query procedures for their associated facts.

The action of a rule is restricted to a single data base assignment. The value to be asserted may be a constant, the value of a datum, or the result of a function evaluation. However, any input/output performed by such a function is beyond the control of Director. No query procedure is automatically associated with the fact mentioned in the action of a rule.

2.2 Interpreter

Director uses both forward and backward chaining. When a fact is given to the system, all possible inferences from the data in the current data base are made using forward chaining. This means that full consideration is given to the newly entered facts. Forward chaining, then, promotes a focus of attention according to the facts offered to Director by its user. When a user query is received, Director establishes a goal, a hypothesis, to confirm or reject. If the goal is not satisfied by simply examining the data base, backward chaining occurs. The backward chaining is guided by a heuristic that tries to maintain focus of attention according to both the user query and the facts recently mentioned (see Section 4.1). During backward chaining additional data may be input, and forward chaining is performed to determine all inferences of this new information. This control structure allows Director to shift focus and goals in response to the user's change of focus and goals.

3. IMPLEMENTATION

This section presents a description of our implementation of Director. We feel that the efficiency of this implementation is important, for it provides a system that is both responsive and functionally helpful.

3.1 Maps For Rule Selection

During rule selection the interpreter must know which facts are contained in the left- and right-hand sides of the rules. This information can be obtained by searching the rule set. Naive searches, however, could be expensive computationally and could make the response time of the system unreasonable. To make this searching efficient, Director maintains two maps. These maps are the rules-add-fact map (RF), and the facts-used-by-rule map (FR). The RF map provides pointers from each fact to the rules that can add it to the data base. The FR map provides pointers between each rule and the facts contained in its left-hand-side, thus specifying which facts have to be true in order for that rule to fire. The maps are built up during a preprocessing stage, which has to be performed only once for a given set of rules.

3.2 The RF Map

Suppose that fact C is in the right-hand-side of rule r: *(If $(A \wedge B)$ then C)*. The RF map entry for this fact would be *$(r <\!\!-\!\!-\!\!> C)$*, indicating that rule r adds fact C to the data base. The information in this map is used during the rule selection portion of the backward chaining phase. For example, if Director is trying to solve goal C, then the RF map provides efficient access to r. This map also allows Director to suppress the firing of certain rules: After a value is assigned to a fact, the system checks the RF map and tries to mark those rules that would assert the same value of this fact. (Rules are not evaluated during the marking process, hence the only rules marked are those that reference this fact by a constant name and assert the same value as a constant.) The marked rules are not evaluated, thus avoiding rule evaluation and the superfluous queries to the user that these evaluations might cause. Suppose, in this example, that rule r assigns a value v to C, which is the present value of C; rule r will be not be selected for evaluation.

3.3 The FR Map

Similarly, the facts-used-by-rule map would contain an entry for rule r, *(A,B<-->r)*, indicating that rule r depends on facts A and B. The map would also contain entries for A and B showing that rule r requires their values in order to be evaluated. When some fact A is added to the data base, all those rules that use A in a forward chaining inference are readily found. In the present example, if A and B are in the data base, rule r is found in the FR map to be usable for forward chaining. This map is also used in the rule selection process of backward chaining. Having determined that rule r will be used to infer a needed fact C, the system readily determines that facts A and B need to be known.

4. QUERIES AND FOCUS

We want Director to select rules for evaluation in a way that minimizes the number of queries posed to the user. Unfortunately, given no semantic information, the problem of choosing the best rules is NP-complete. To avoid this complexity, we try to suppress unnecessary rule execution and choose among the eligible rules with a heuristic. Our heuristic for selecting which rule to evaluate is based on natural language considerations.

4.1 Heuristics for Backward Chaining

Suppose the knowledge base of an expert system contains the following set of rules:

1. IF A THEN B
2. IF Y AND B AND Q THEN D
3. IF Y AND B AND W THEN D

Also suppose the user of the system issues a query which adds facts A and Q to the data base and states that the goal is to know D. Director first forward chains to make all the possible inferences given the contents of the data base. In this case rule 1 is evaluated, adding B to the data base. Now the system backward chains. We want Director to pick the next rule in such a way as to guarantee the most focused conversation. To promote this behavior, Director tries to select the rule with both the goal in its right-hand-side and the greatest number of facts most recently added by the user in its left-hand-side. This implies that Director must differentiate those facts derived by rules and those entered by the user. Furthermore, Director assigns a time-stamp to each fact added by the user. In the example Director would try rule 2 first, because it contains Q which was entered by the user. However, if Q were unknown, the system would choose arbitrarily between the rules 2 and 3. This heuristic is guided by the facts entered recently and thus does not always give optimal behavior. However, if the user mentions relevant facts as he issues queries (as would be expected if the user understood the problem domain), the behavior should be quite natural, giving a focused conversation.

4.2 User Control for Backward Chaining

Sometimes the user of Director has semantic knowledge of the queries and can, therefore, better direct the selection of rules. Forward chaining can be controlled simply by the facts that are added to the data base. Backward chaining can be controlled by the facts and the queries issued to Director. An optional mechanism is provided in Director to allow the user to help direct the rule selection process. Normally, expert systems use only information in the right-hand-sides of their rules to initiate backward chaining. Director can also use information in the left-hand-side of rules when selecting rules to use as a starting point of the backward chaining process. If the user of Director supplies this left-hand-side information when making a request, it will be used in the initial rule selection. For example, consider the following set of rules:

1. IF A AND B THEN D
2. IF A AND C THEN D

Suppose a user issues the following query:

"Given B, is D?"

Using the maps, Director identifies that the general goal is implied by rules 1 and 2. The system now selects a rule as the starting point of the backward chaining process, choosing rule 1 according to user control. The next section describes the mechanism provided for allowing the user to make such a focused query. We have in mind that the "user" will generally be a semantic module that translates user questions into requests to Director.

5. SELF DESCRIPTION

So far, we have described what we call Director's inferencing function. The system has another function, called display. There are many instances when a user wants the system to provide information without providing inferencing. For example, a user may want to see everything the system knows about a certain fact A and issue the following request: "Tell me about A." Our system can handle a query of this sort by using the maps. All the rules that contain A in the left-hand-sides are found with the help of the FR map. Similarly, all the rules containing A in the right-hand-side are found with the help of the RF map. All rules containing A are returned as a response to the above query. Providing this information is done quickly because Director does not perform inferences, but rather only references the maps.

In the future, we would like to build a semantic module (that uses Director) to handle queries of the form "Do A and B imply C," where A and B are on the left-hand-side of rules, and C is on the right-hand-side. If A, B and C are all in the same rule, the query is answered simply by returning the rule. However, if A,B and C are not in the same rule, the response to this query could be expensive to compute because it involves tracing all the paths from A and B to C. Using Director all of this tracing is done with the maps only, which requires less computation than performing inferences for a similar query. The mechanism needed to handle a query such as the above, as well as other queries, is currently being developed.

6. CURRENT STATUS AND CONCLUSIONS

Some extant interpreters (e.g., in KEE [In 85]) could be modified to provide the characteristics of Director. Indeed, our effort can be viewed as an outline of the modifications required. However, we felt the cost of such alterations would be too large in our environment, thus we are implementing Director in Zetalisp with rules that are defined using the local tools, Describe [Bl 84] and Portal. We are using Director to build portions of a small expert system called Taxpert [En 85], which deals with personal income tax matters. The system contains a number of agents, which cooperate to solve an assortment of tax problems. Some of the agents gather information, some fill out tax forms, and others give advice. Director serves as the control mechanism for the dependency expert, which contains about 50 rules.

The inference engine we have described uses a combination of forward chaining and backward chaining. It also makes available some descriptions of its rule base and allows for a limited form of user control over its backward chaining mechanism. This facility allows the user to ask questions about the information contained in the rules but not normally supplied by expert systems. Director facilitates construction of rule-based systems with natural language interfaces, presenting focused dialogue and reducing the number of user queries. This interpreter is also useful in other domains, where the decisions have to be made quickly, or where user queries are expensive.

References.

[Bl 84] Blumenthal, R.L., Dickinson, A., Ensor, J.R., and Joseph, R.L., "Describe -An Explanation Facility for Object-based Expert Systems", in preparation.

[Bo 83] Bobrow, D. G., and Stefik, M., "The LOOPS Manual," Xerox Corp., Palo Alto, CA, 1983.

[Cl 83] Clancey, W.J., "The Epistemology of a Rule-Based Expert System - a Framework for Explanation," *Artificial Intelligence*, Vol. 20, pp. 215-251, 1983.

[Da 84] Datskovsky, G., "Menu Interfaces to Expert Systems: Evaluation and Overview," Technical Report CUCS-168-84, Columbia University, New York, NY, 1984.

[Da 85] Datskovsky, G., "Natural Language Interfaces to Expert systems," Technical Report CUCS-169-85, Columbia University, New York, NY, 1985.

[Da 78] Davis, R., "Knowledge Acquisition in Rule-Based Systems-Knowledge About Representation as a Basis for System Construction and Maintenance," in *Pattern Directed Inference Systems*, Academic Press, New York, NY, 1978.

[Du 79] Duda, R.O., Gaschnig, J., Hart, P.E., "Model Design in the PROSPECTOR Consultant System for Mineral Exploration," in D. Michie (ed.), *Expert Systems in the Micro-Electronic Age* , pp. 153-167, Edinburgh Univ. Press, 1979.

[Du 84] Duda, Richard, Presentation at the IEEE Workshop on the Principles of Knowledge-Based Systems, Denver, CO, 1984.

[En 85] Ensor, J. R., Gabbe, J. D., and Blumenthal, R. L., "Taxpert – A Framework for Exploring Interactions Among Experts," in preparation.

[Fo 81] Forgy, Charles L., "OPS5 User's Manual", Carnegie-Mellon University, Pittsburgh, PA, 1981.

[Gr 77] Grosz, B.J., "The Representation and Use of Focus in a System for Understanding Dialogs," *Proc. of the Fifth IJCAI*, Vol. 1, pp. 67-73, 1977.

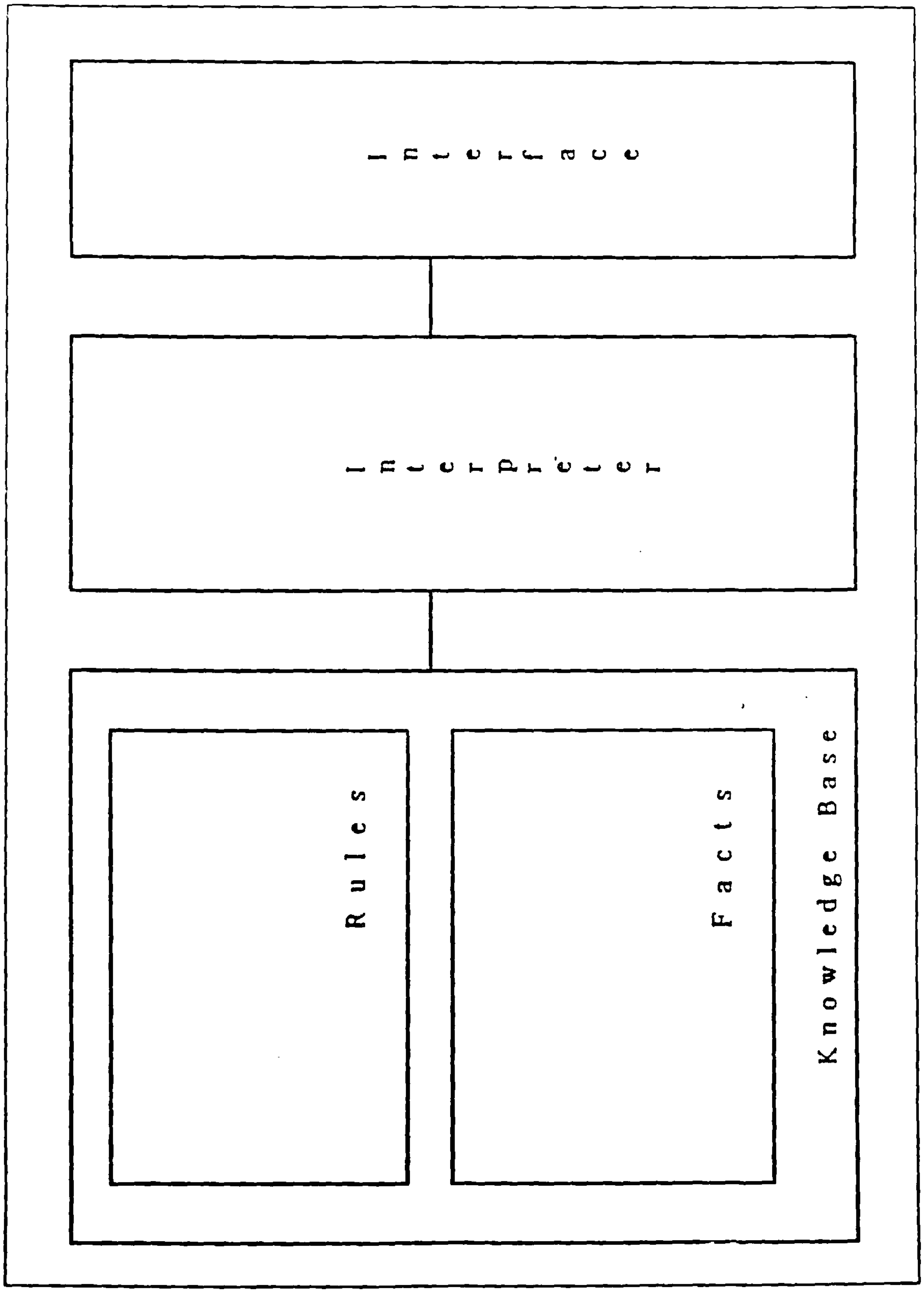
[Ha 85] Hayes Roth, F., "Rule Based Systems," *Comm. of the ACM*, Vol. 28, No.9, pp. 921-932, September, 1985.

[In 85] IntelliCorp, "KEE 2.1 Software Development System Reference Manual."

[Po 82] Pollack, M., Hirschberg, J., and Webber, B., "User Participation in the Reasoning Process of Expert Systems," *Proc. of National Conference of AAAI*, pp. 358-361, 1982.

[Po 83] Pollack, M.E., "Generating Expert Answers Through Goal Inference", Technical note, SRI International, Menlo Park, CA., 1983

[va 81] van Melle, W., *et al*, "The Emycin Manual," Technical report STAN-CS-81-885, Stanford University, Stanford, CA, 1981.


Interface
Interpreter
Rules
Facts
Knowledge Base