Rule-Based Inference in Large Knowledge Bases *

William Mark

USC/Information Sciences Institute

ABSTRACT

The process of inference can be conceived as the continual redescription of a given structure in the system knowledge base until a desired structure is formed. Inference rules then represent single transformations to be used in this redescription process. If the rules are themselves integrated into the knowledge base, the rule application mechanism can take advantage of knowledge base organization to maintain an efficient inference process, even in systems that must deal with large amounts of knowledge. This paper presents a methodology for implementing rule application, and gives two examples of its use in knowledge-based systems.

I GROWING ASPIRATIONS

Having gained some experience with knowledge-based systems (e.g., [3], [9], [11]), our aspirations are growing. Future systems (for VLSI design, office automation, etc.) will have to model more of the knowledge of their domains and do more interesting things with it. This means larger, more structured knowledge bases and inference mechanisms capable of manipulating the structures these knowledge bases contain. The necessarily large investment in building these systems, and the very nature of some of the applications (e.g., data base query, cooperative interactive systems), also require these systems to be more adaptable than before to new domains within their purview (e.g., a new data base, a new interactive tool).

II RULE-BASED INFERENCE

The need for adaptability argues strongly for perspicuity and modularity in the inference engine: the adapter must be able to see what must be changed (or added) and be able to make the alterations quickly. Inference mechanisms based on rules have these characteristics. Unfortunately, most rule-based approaches rely on small, simply structured system knowledge bases (e.g., the rule-based formalism used in [10] and [4] is dependent on representation in terms of triples).

As rule-based systems grow to encompass a large number of rules, and as they are forced to work on complex knowledge structures to keep pace with modern knowledge base organizations, two major problems arise:

- The inference mechanism becomes inefficient: it is hard to find the right rule to apply if there are many possibilities.
- Rules begin to lose their properties of modularity and perspicuity: the "meaning" of a rule, especially in the sense of how it affects overall system behavior, becomes lost if the rule can interact with many other rules in unstructured ways.

The remainder of this paper describes an approach to solving these problems based on a philosophy of doing inference that is closely coupled with a principle of rule base organization. This approach is discussed in the context of two implementation technologies.

III PHILOSOPHY

The inference methodology described here is a reaction to the use of rules as pattern/action pairs in a rule base that is not intimately related to the program's knowledge base. The basic philosophy [6] is to treat expert system inference as a process of redescription: the system starts with some kind of problem specification (usually a single user request) and redescribes it until it fits a known solution pattern (a data base access command or an implemented program function in the two examples described below). The control structure for this redescription depends on a knowledge representation that explicitly includes the inference rules in the knowledge base.

The redescription effort proceeds in two modes: a narrowing-down process that simply uses any applicable rules to redescribe the input as something more amenable to the program's expertise; and a homing-in process that

* This research was supported in part by the Defense Advanced Research Projects Agency under Contract No. DAHC15 72 C 0308, ARPA Order No. 2223, and in part by General Motors Research Laboratories. Views and conclusions contained in this paper are the author's and should not be interpreted as representing the official opinion or policy of DARPA, the U.S. Government or any person or agency connected with them.

takes the description remaining from the first process (i.e., when no more applicable rules can be found), finds the most closely related solution pattern, and then uses that solution as a goal. In this homing-in mode, the inference procedure uses consequent rules to resolve the differences between the current description and the desired end.

The narrowing-down phase focuses solely on the input and transforms it into something closer to the system's solution model. Depending on the input, the result of narrowing-down might be an actual solution (meaning the input request was quite close to the system expectations), or something that the system has no idea of how to deal with (meaning that it was very far from what was expected). Given that the user has *some* idea of what the system can do, and that he wants to make himself understood, we can assume that the result of narrowing-down will often be only a minor perturbation of a system solution. This enables the homing-in process to find a closely related solution to use as a goal. Very specific consequent rules can then be used to resolve the few remaining differences.

Assuming rule-based inference and a structured system knowledge base, the above philosophy can be restated as follows:

- o Inference is the transformation of one significant knowledge base structure into another.
- o The organization of the knowledge base organizes the rule base.
- o The control structure supports both straight rule application anywhere in the knowledge base and consequent rule application in the context of a well defined goal.

This approach makes rule application more efficient, even in a large knowledge base, because highly specific rules are not used--are not even looked for--until they have a good chance of success. That is, the system does not have to look through everything it knows every time it must apply a rule. This efficiency is further enhanced by the fact that inference is modeled as redescription and rules are tied closely to the system knowledge base. The inference mechanism will not even look at rules that do not relate directly to the kind of knowledge structures found in the description to be transformed. This makes the problem of finding the right rule to apply far less dependent on the number of rules in the system. The highly structured and detailed nature of the knowledge base now works for efficiency of rule application rather than against it.

Separating the inference process into modes and tying rules directly to the structures of the knowledge base also enhances the modularity and perspicuity of the rule base, making it easier to adapt the system to new domains. It is easier to see the meaning of an individual rule because it has a specific function in the inference process: either a fairly general narrowing-down transformation that does not depend on the application of other rules, or a quite specific, highly interdependent homing-in transformation that is used only in those bounded redescription situations in which both end-points (given description and goal description) are known. Furthermore, since all rules are an integral part of the knowledge base, the meaning of a rule with respect to other rules can be determined by its relationship to those rules in the knowledge base. The interpretation of inference as redescription means that this static relationship between rules in the knowledge base corresponds directly to the dynamic relationship during the inference process. That is, because the sole action of a rule is redescription, its behavior with respect to other rules can be completely determined by comparison of the condition and conclusion knowledge structures of the rules involved.

IV TWO APPROACHES TO IMPLEMENTATION

This methodology has been used to implement the inference components of two knowledge-based systems using quite different technologies. The first, a pattern-match design, is simply sketched, while the second, a network based scheme, is presented in more detail.

A. Inference in Askit

The Askit system currently being developed at General Motors Research Laboratories ([7], [8]) is a natural language data base query facility designed to be adaptable to new domains (e.g., a new project scheduling data base or a new inventory management data base). The inference task is to translate the user's query into the appropriate set of data base commands to provide the needed data. Askit's knowledge base consists of case structures representing user request types, data base commands, individual English words, etc. Rules are expressed as transformations between system case frames. The condition part of the rule is a partially instantiated case frame that is treated as a pattern to be further instantiated (i.e., "matched") by a description (a fully instantiated case frame) from the program's current state. The conclusion is a case frame to be filled on the basis of the instantiation of the condition part. When the rule is applied, the instantiating description is replaced in the current state by the new structure generated from the conclusion part of the rule. Rules therefore represent allowable redescriptions of case frames for certain choices of case fillers.

For example, the following is a rule for redescribing restrictions in certain user requests as SUBSET commands to the database system:

```
(REQUEST (OBJECT <table-data>:obj)
         (RESTRICTION =obj (<column-data>:property
                            :relation :value))
->  (SUBSET =obj WHERE =property =relation =value)
```

That is, the condition of the rule applies to any request which deals with something that is classified as "table data" in the system knowledge base, and which expresses a restriction of that table data in terms of "column data". The conclusion states that such a request can be redescribed as a SUBSET command. When the rule is applied, the SUBSET command replaces the request in Askit's current state.

Rules are organized into "packets" based on the case frames in the knowledge base: all rules whose conditions are partial instantiations of the same case frame are grouped in the same packet. For example, all rules that deal with restrictions in user requests would form a packet. The packet is represented by a packet pattern which states the common case structure dealt with by the rules of the packet, i.e., a generalization of their condition parts. The packet pattern for the "restriction in requests" packet would be:

```
(REQUEST (OBJECT :obj) (RESTRICTION =obj :rstr))
```

Packets play a key role in Askit's rule application process. The process begins by matching a description in Askit's current state against the packet patterns known to the system. If a match is found, the individual rules of the packet are tried. If one of the rules is successfully matched, the input structure is redescribed and placed back in the current state.

If no rule matches, Askit goes into homing-in mode. It posits the discrepant part of a not-fully-matched rule from the packet as a new description in the current state, and looks for consequent rules to match this description. Consequent rules are also organized into packets (but based on their conclusions rather than their conditions). If Askit finds a matching rule in the consequent packet, and if the condition part of the consequent rule matches other descriptions in the current state, the discrepancy between the original description and the partially matched rule is considered to be resolved, and processing continues. Otherwise, other possible discrepancies or other partially matched rules are posited, and homing-in is tried again.

Thus, the narrowing-down process in this system is represented by normal packet application, in which an initial description is matched against packets and successively redescribed until it can be seen as an instantiation of one or more system case structures representing data base commands. If this process breaks down, i.e., if a packet pattern is matched but no rule in the packet can be matched, the input description is treated as a perturbation of one of the case structures represented by rule conditions in the packet. Consequent rules are then used to home-in the discrepant description on the desired case structure. Successful resolution of the discrepancies allows normal processing (i.e., narrowing-down) to continue.

B. Inference in Consul

The Consul system is being designed to support cooperative interaction between users and a set of online tools (for text manipulation, message handling, network file transmission, etc.). This "cooperative" interaction includes natural language requests for system action and explanation of system activities. Since user requests may differ radically from the input form expected by the tool designer, Consul's inference task is to translate the user's natural form of requesting system action into the system's required input for actually performing that action.

In the Consul system, the dependence of rule representation and organization on the system knowledge base is carried much further than in Askit. On the other hand, the control structure does not have to be as complex. Consul's knowledge base is a KL-ONE [2] representation of both tool-independent and tool-dependent knowledge. The major organizational framework of the knowledge base is set by the tool-independent knowledge, with the tool-dependent elements instantiating it.

Inference in Consul is a process of taking an existing description in the knowledge base, redescribing it in accordance with an inference rule, and then reclassifying * it in the knowledge base. If the description can be classified as the invocation of an actual tool function in the system, then the description is "executable", and inference is complete. A key aspect of Consul inference is that inference rules are also represented in KL-ONE in the knowledge base; the same process of classification that determines the status of a description also determines the applicability of a rule (compare [5]).

For example, let us examine Consul's treatment of the user request

"Show me a list of messages."

Parsing (using the PSI-KLONE system [1]) results in the structure headed by **ShowAct.1** in figure 1. Consul classifies this structure in its knowledge base, finding, as shown in figure 1, that it is a subconcept of the condition of **Rule1**. This means that Consul can redescribe the request **ShowAct.1** as a call to a "display operation" according to the conclusion of **Rule1**. The result is a new description, **DisplayOperationInvocation1.1**, which Consul then classifies in its knowledge base. If, via this classification process, the new description is found to be a subconcept of an executable function, inference is

* The classification algorithm was written by Tom Lipkis.

complete--DisplayOperationInvocation1.1 can simply be invoked by the Consul interpreter. Otherwise Consul will have to use additional rules to further refine the description until it can be seen as an actual call on some other function or functions.

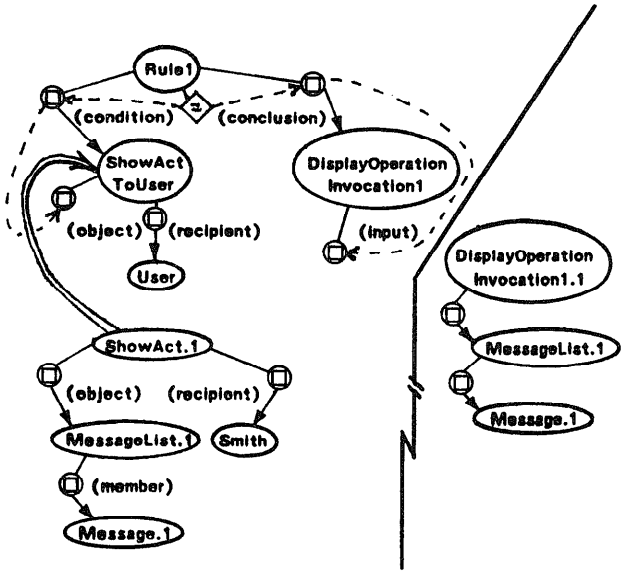

Figure 1: Rule Application in Consul

Figure 2 shows the classification of DisplayOperationInvocation1.1 in Consul's knowledge base. Unfortunately, the system does not know very much about it in this position: it is not a subconcept of an executable function (shown shaded), nor is it a subconcept of any rule condition. Since no applicable function or rule can handle the current description, the system seeks a "closely related" function description to use as a mapping target. The definition of "closely related" is that the two descriptions must share a common ancestor that is a "basic" concept in Consul's actual knowledge base--i.e., the ancestor cannot be part of a rule itself, nor a newly generated description. Referring to figure 2, the function description DisplayMessageSummaryInvocation is closely related to the current description DisplayOperationInvocation1.1 because they share an appropriate ancestor, DisplayOperationInvocation.

Once a related tool function description is found, it is used as a goal for consequent reasoning. First, Consul must find the discrepancies between the current description and the desired result. These discrepancies are simply the differences that prevent the original description from being classified as a subconcept of the desired description: in this example, those features of DisplayOperationInvocation1.1 that prevent it from instantiating DisplayMessageSummaryInvocation. The discrepancy is that the "input" role of the current description is filled with a list of messages (MessageList.1), while the executable function DisplayMessageSummaryInvocation requires a list of summaries (SummaryList).

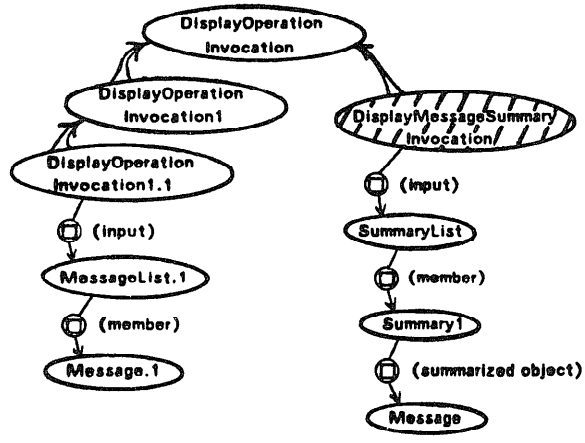

Figure 2: Finding a Closely Related Function Description

Consul must therefore find a way to redescribe a list of messages as a list of summaries if it hopes to see the current description as an instantiation of this particular executable function. There are two ways to transform the current description: rules and executable functions (since functions produce new descriptions via output and side-effects). Rules are preferable because they save tool execution time; Consul therefore looks for rules first. In this case, there is a quite general rule that produces the desired effect. Users frequently ask to see a list of things (messages, files, etc.) when they really want to see a list of *summaries* of these things (surveys, directories, etc.). Consul therefore contains a rule to make this transformation, if necessary.

The rule, shown as Rule2 in figure 3, says that if the current description is a display operation to be invoked on a list of "summarizable objects", then it can be redescribed as a display operation on a list of summaries of those objects. Consul finds this rule by looking in the knowledge base for rules (or executable functions) that produce the "target" part of the discrepancies found earlier. As shown in figure 3, Rule2 can be found by looking "up" the generalization hierarchy (not all of which is shown) from SummaryList.

Consul must next be sure that the condition part of the rule is met in the current state of the knowledge base (this includes the current description, descriptions left by previous rule applications and tool function executions, and the original information in the knowledge base). If the condition is not satisfied, Consul will try to produce the needed state through further rule application and function

execution--i.e., through recursive application of the consequent reasoning process.

In this example, the condition of **Rule2** is entirely met in the current description (see figure 3). Furthermore, the conclusion of **Rule2** resolves the entire discrepancy at hand. In general, a combination of rules and functions is needed to resolve discrepancies. Therefore, with the application of **Rule2**, Consul has successfully redescribed the initial user request as an executable function. The inference process passes the resulting description on to the interpreter for execution, and the user's request is fulfilled.

Narrowing-down and homing-in are much the same in Consul as they are in Askit. In Consul, however, relationships between rules and notions such as "perturbation" and "closely related structure" come directly from the existing knowledge base representation; a superimposed packet structure is not necessary. The control structure of rule application therefore need not consider the separate issues of matching packets, setting up packet environments, matching rules, etc. Instead, classification includes matching, as applicable rules are found "automatically" when a newly generated description is put in its proper place in the knowledge base. Finding related knowledge structures and consequent rules are also classification problems, representing only a slightly different use of the classification algorithms.

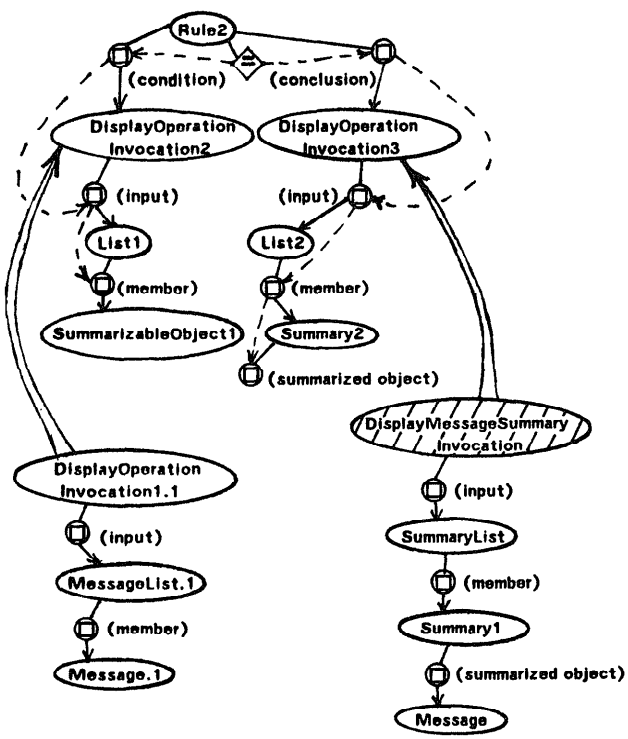

Figure 3: Using Consequent Reasoning

[1] Rusty Bobrow and Bonnie Webber, "PSI-KLONE: Parsing and Semantic Interpretation in the BBN Natural Language Understanding System," in *Proceedings of the 1980 Conference of the Canadian Society for Computational Studies of Intelligence*, CSCSI/SCEIO, 1980.

[2] Ronald Brachman, *A Structural Paradigm for Representing Knowledge*, Bolt, Beranek, and Newman, Inc., Technical Report, 1978.

[3] Bruce Buchanan, *et al.*, *Heuristic DENDRAL: A Program For Generating Explanatory Hypotheses In Organic Chemistry*, Edinburgh University Press, 1969.

[4] Randall Davis, *Applications of Meta Level Knowledge to the Construction, Maintenance and Use of Large Knowledge Bases*, Stanford Artificial Intelligence Laboratory, Technical Report, 1976.

[5] Richard Fikes and Gary Hendrix, "A Network-Based Knowledge Representation and its Natural Deduction System," in *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, IJCAI, 1977.

[6] William Mark, *The Reformulation Model of Expertise*, MIT Laboratory for Computer Science, Technical Report, 1976.

[7] William Mark, *The "Askit" English Database Query Facility*, General Motors Research Laboratories, Technical Report GMR-2977, June 1979.

[8] William Mark, *A Rule-Based Inference System for Natural Language Database Query*, General Motors Research Laboratories, Technical Report GMR-3290, May 1980.

[9] William Martin and Richard Fateman, "The MACSYMA System," *Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation*, 1971.

[10] Edward Shortliffe, *MYCIN: Computer-Based Medical Consultations*, American Elsevier, 1976.

[11] William Swartout, *A Digitalis Therapy Advisor with Explanations*, MIT Laboratory for Computer Science, Technical Report, February 1977.