This paper describes a system designed to help a user of a rule-based system understand the discrepancy that exists when a belief that is held or hypothetically considered by the user is not concluded by the system. Such discrepancies can arise in situations such as 1. user responses to intelligent tutoring systems, 2. justification of the conclusions of expert systems, 3. the use of negative cases during knowledge engineering, and 4. hypothetical or counterfactual reasoning in an interactive intelligent system. The system utilizes the structures of rules and proofs rather than semantic knowledge or discourse structure, and consists of three mechanisms: backward tracing through rules which plausibly could have concluded the mistaken belief, analysis of failures of complex rule matches, and explanations of the structure of the proof structure.

1. Introduction: the importance of meta-reasoning

Many of the problems which have impeded the more rapid acceptance and dissemination of knowledge-based systems and techniques can be subsumed under the general issue of the brittleness of reasoning. Naive parties expect more in the way of robustness, common sense reasoning and reasonableness, and, in particular, explanation and interactive capabilities, than the present generation of systems can offer. The problem of brittleness has lead some to abandon the central insight of declarative knowledge representation for the allure of approaches such as connectionism and reactive systems which are, for general problem solving at least, computationally intractable and demonstrably inadequate. Others (e.g., Lenat & Guha, 1990) have sought solutions in attempts to capture massive amounts of general knowledge and/or in fine grained solutions to the problems of knowledge representation. These
semantic, knowledge-based approaches are well motivated and in the long term necessary, but they are generally not applicable in the short term to the current generation of expert systems, nor are they sufficient in themselves to overcome the problems of current systems.

What is generally also required is the ability to reason about the reasoning process itself, i.e., meta-reasoning. The ability to reason about reasoning is critical for a wide range of important issues that involve communication and understanding between humans and intelligent systems, including: plan-based cooperative and competitive interaction in multi-agent intelligent systems, plan-based language understanding, user modeling and understanding user replies in tutoring dialogues, the use of hypothetical or counterfactual reasoning and, in the field of knowledge engineering, the development of expert system interfaces and knowledge acquisition tools that support dialogues concerning the form and limitations of the system's knowledge and reasoning capabilities, including dialogues regarding negative cases and hypothetical or counterfactual reasoning. (Metzler, 1990) Although many reasoning paradigms in artificial intelligence involve models of aspects of meta-reasoning, (e.g., case-based reasoning, explanation-based learning, and plan-based understanding), it is difficult to develop a general model of meta-cognition. This paper describes a relatively simple approach to one aspect of meta-reasoning which has potential applicability to a variety of different kinds of systems and types of reasoning.

2. Interaction, question answering and explanation

Question answering and explanation are critical for the acceptance of expert systems. Even stand-alone traditional expert systems must be able to justify their conclusions to a user, and knowledge engineers must be able to trace a system's line of reasoning to compare it with the human expert's and otherwise insure its soundness. Unfortunately, the rule traces of traditional expert systems, while providing more meaningful information than do conventional systems, do not provide all of the kinds of information a user or developer might want. For instance, they do not provide deep answers regarding the reasons for the rules themselves.

Systems that support cooperative human machine problem solving, and multi-agent "distributed" intelligent systems introduce additional problems. For instance, there are issues regarding the interpretation of references to specific or general objects, as in the interpretation of a mouse click in terms of the various single or grouped objects on the
screen that it might refer to, or the interpretation of a request for a
definition in a particular context. Although these issues of reference and
explanation have often been addressed in theories of linguistic
interpretation, the issues present themselves in any form of
communication whether or not it involves natural language.

In addition to questions regarding reference, another general category of
questions and explanations that is common in interaction and essential to
supporting interaction in intelligent systems is that of questions and
explanations regarding the reasoning process itself; how conclusions were
reached, why lines of reasoning were rejected, how conflicts were
resolved and so on. For instance, perhaps the most common question that
clients ask human experts (e.g., consultants) is the negative "why didn't
you" ... (recommend, try, think etc.), rather than the positive "why did
you..." The conventional rule trace capabilities of expert systems answer
two forms of positive questions, those most often referred to as "how"
questions which have to do with how the system reached a particular
conclusion, and those, most often referred to as why questions, which
address why the system wants a certain piece of data. The system
described here addresses the "why not" form of questioning.

3. The Mistaken-Belief Query System (MQS)

MQS is designed to explain why an assertion is not believed by a rule-
based reasoning system. It is aimed at explaining plausible or "near-miss"
mistaken beliefs such as might arise as responses to an intelligent tutoring
system's queries (Katz et al., 1989) or as the "why not" sorts of questions
that users or developers might want to ask of an expert system (Metzler,
1989). As a development tool, this capability allows a knowledge
engineer to see how the system is handling near-miss negative cases, and
to ensure that it is rejecting likely mistakes for the right reasons. For end
users, it can ensure a greater degree of confidence in the conclusions and
in the line of reasoning provided by an expert system. More generally, it
provides part of the capability to reason counterfactually or hypothetically which is necessary to support interactive problem solving
between multiple intelligent agents.

Although the term "mistaken belief" might appear to imply the
assumption that the user is wrong and the system is correct, it is more
accurate to say we are looking to explain why the user and the system
disagree, since even in the cases of tutoring systems or mature expert
systems, we would want to allow for the possibility that the user is
correct and the system mistaken. In the case of interactive intelligent systems, where the human user might be presumed to have judgment, responsibility and knowledge that at times exceeds that of the system, we might even want the user to be able to direct the system to proceed on the presumption that the user is correct, particularly after the user has been able to ascertain why the system has come to a conflicting conclusion. In the case of knowledge engineering tools, we would want to go even further, in that we would want the developer to be able to change the system so that it would agree with the user in the future.

The current version of MQS is embedded in Entrypaq, a commercially available expert system shell which is itself implemented in HyperCard. Entrypaq has simple forward and backward chaining rules and a basic frame system. The clauses of the rules refer to <instance slot value> triples, corresponding to individual predications, i.e. slot(instance value), rather than frames, objects or working memory elements. Matching against instances of objects with combinations of properties is achieved by using several clauses that refer to the same instance, i.e., that have the same variable or constant in the instance position of the <instance slot value> triple. Most of the source code that implements Entrypaq is available as HyperCard scripts and can be altered and utilized to implement other systems. The current version of MQS consists of additions to the scripts of Entrypaq, as well as new HyperCard cards, backgrounds, scripts and handlers, and use is made of existing calls in Entrypaq, e.g., to frame and rule editors, and the backward-chainer, whenever possible. The forward-chainer was rewritten to allow for multiple "isa" or object variables since Entrypaq allows only one "isa" variable in the antecedents of a rule, (all other variables must be bound by the time the pattern matcher reaches them).

MQS takes as input an <instance slot value> fact, which is presumably false, (at least from the system's point of view) and tries to help the user understand why the system does not believe this fact. The assumption is that if it is a plausible mistake it would be useful to see how close the system comes to deriving the same conclusion. Since it is likely that the user, in coming to a mistaken conclusion, has used an incorrect variation of a correct rule, or has mistakenly applied a correct rule because of mistaken beliefs regarding facts, showing how the system rejects potential supporting lines of reasoning is likely to reveal the user's mistaken line of reasoning, or something close enough to it to allow the user to figure out why (or whether) he or she is wrong. The explanatory reasoning consists of three components. The first is essentially a trace (or, more accurately,
a "browse," since these rules had not actually fired) backward through rules which would have concluded the mistaken belief (but didn't since they never fired). The second is an attempt to explain in detail the lack of a complex match for a single rule. The third is an attempt to explain some of the structure of the (failed) proof structure.

3.1 Backward trace through relevant rules.

When a user believes an assertion to be true, the basic premise of this system is that the user will frequently have a reason or reasons to believe it to be true, and that whether the user is a tutoring system student, or the user or developer of an expert system or an interactive intelligent system, it can be valuable for the user to compare his or her reasons with the reasoning of the system. It is assumed that in many useful cases, the user will discover either mistaken beliefs that interfere with the application of correct rule knowledge, or mistaken versions of the correct rules. The mistaken beliefs regarding facts or rules may ultimately turn out to be either those of the user or those of the system.

When MQS is given an assertion to explain, and it has determined that the assertion is not believed or provable, it proceeds to search for rules whose conclusions match the assertion. Since the assertion can not be proven, any rule matching the assertion must fail for some reason. The simplest cases are when rules fail because individual premises fail. Each rule found is displayed in a window showing the conclusion and each of the premises. The backward-chainer is invoked separately for each premise, and a separate window displays the false premises that blocked the rule. When a premise is false because the knowledge base currently contains a direct contradiction, that contradiction is displayed in parentheses below the false premise. When a premise fails without a direct contradiction, it means that the backward-chainer failed to prove it.

At this point, this display can be interpreted in a number of ways and MQS gives the user several choices. One is to ask to see another rule whose conclusion matches the original assertion. Such a "horizontal" move through the space of possible lines of reasoning might be made if the user had in mind a particular line of reasoning which did not correspond to the rule currently displayed. It could also be useful for general browsing of the reasoning of the system in regard to the assertion in question. On the other hand, the currently displayed rule might be closely related to the line of reasoning the user was following. If the user believed a different version of the rule, e.g., one with fewer antecedent clauses or looser
restrictions on one or more elements in the antecedents, the comparison
between the system's version and the user's version of the rule should be
quite informative in many cases. (Even in the case in which the user
might be forced to go to an outside source of expertise for a full
explanation of the discrepancy, the user can do so with a very focused
question.)

In other cases, the user might agree with the system's rule knowledge,
and, in particular believe that the currently displayed rule instantiation is
plausible, but fail to understand why one or more of the blocked premises
of the rule were not believed by the system to be true. In these cases the
user can move "vertically" (recursively) down into the reasoning space by
selecting one of the false premises of the displayed rule and recursively
invoking the MQS mechanism for this assertion. As such horizontal and
vertical moves are made, the system records each rule instantiation
visited as a structure (a HyperCard card with appropriate fields), and
records pointers between them to represent the moves taken. The system
allows the user to traverse the resulting graph, e.g., by returning to
previously displayed rule instantiations to check out another false
premise. Thus, in the case of a tutoring system, a user can readily look for
aspects of the system's reasoning which closely match the users reasoning,
and in the case of interactive systems or knowledge engineering tools, a
user can readily gain an appreciation for the range of reasoning the
system is capable of regarding the assertion in question.

3.2 Explaining complex match failures.

The reason for the premises of a rule not matching may not be as simple
as individual false premises, but may involve the mutual constraints
between the premises. This situation is revealed when the system finds a
rule that matches the assertion in question, which it has already
determined to be not provable, and yet finds that all of the premises of
the rule are individually provable. At this point we want the system to
provide a useful description of how the constraints are operating to
prevent a consistent matching of all of the premises. Exactly how to do
this depends on the detailed form of the reasoning system, e.g., on
whether the premises contain assertions or objects, and the form of
predicate tests allowed. In the case of a system whose premises are
assertions, a major first step in this process is to explicitly identify the
object descriptions that are composed of the individual assertions and see
whether objects matching these descriptions exists. MQS does this by
creating an individual temporary rule for each object. Each such rule is
composed of only the premises from the rule in question that refer to the same object instance. Each such temporary rule is given to the backward-chainer, and if such a rule fails, it means that there is no such object that meets the internal constraints pertaining to the object itself. If all such temporary object rules can be satisfied by the backward-chainer, but the original rule can not, it means that the constraints that are blocking the satisfiability of the entire rule have to do with the relationships between the objects the rule refers to. It may be enough in practice to point this fact out, perhaps by graphically displaying the variables mentioned in more than one object which carry cross-object constraints, but we are exploring ways to be more explicit about this information. One strategy is to create temporary rules which substitute new variable names for some of the instances of cross-object variables, thus breaking the constraints at particular points. For instance, if there are two uses of cross-object constraints and breaking one of them allows the rule to go through and breaking the other doesn't help, the problem has been localized in the first constraint.

3.3 Explaining failed proof structures.

In some cases, a useful explanation may require even more than this. It may involve, for instance, the realization that in the case of a cross-object constraint problem, that the constraint problem is more with one of the objects than with the other. This might be because one object description matches a smaller number of domain objects than does the other, or because the range of values taken by the cross-object variable is smaller for one object description than for the other. Although how exactly to interpret this, and what to do about it, are semantic and domain specific questions, we believe that such syntactic hints regarding where matching problems lurk can be useful tools.

4. Future work

The following sections briefly describe several issues we are currently considering. These issues require greater programming power and flexibility than is afforded by HyperTalk, and we are rewriting the current system in Harlequin's KnowledgeWorks (a general programming environment combining Common Lisp, CLOS, an OPS compatible forward-chainer and a Lisp implementation of Prolog), in order to address these issues.¹

¹ It is interesting to note that the Harlequin KnowledgeWorks rule browser tool has facilities similar to some of what is described in sections 3.1 and 3.2 above. For a
4.1 Consequent Match Failures

The current methods concentrate on matching problems involving the antecedent clauses of a forward chaining rule-based system. It is possible that a mistaken user conclusion is based on an incorrect notion of what sort of belief building or belief changing action might be taken on the consequent side of a rule. In this case, the rule that would be informative to display would be one which has a consequent similar to, but not matching directly with the user's "mistaken" belief. Finding such rules involves developing a general notion of approximate matching appropriate for this sort of system.

4.2 Conflict Resolution Based Failures

In some situations and rule-based system architectures, a fact or action might be blocked even though a rule is available to correctly reach that conclusion because the system's conflict resolution strategy prefers a different rule instantiation. Common examples of this would be a preference for a more specific rule instantiation over a more general one or a preference for a rule instantiation that matches more recent knowledge-base items over an instantiation that matches older knowledge-base items. This is not a problem in a monotonic system where the distinction between true assertions that have been made and those which have not yet been made is not ultimately critical. But in nonmonotonic situations, such as design or planning problems, in which a particular action can block the applicability of other actions which may have be acceptable (although not preferred) up until the point at which the alternative was taken, this problem can be very important. Such situations are not difficult to deal with at the top-level. They are revealed by the discovery of a rule instantiation that matches the "mistaken-belief" and that is, in fact, provable by the backward-chainer, but that has not been asserted by the forward-chainer. In fact, this is equivalent to inspecting the conflict set of a standard forward-chainer.

Browsing backward recursively, as we do for the problem of rules that don't match the knowledge base is much more difficult in this case however. The problem is that the conflict set at a point other than the
given rule, it will show whether each of its antecents is matched and whether all the antecedents are can be matched consistently. It will not search for rules to reach a given conclusion or recursively go back to see why antecedents aren't true, nor will it look for more complex matching problems.
current environment is determined by the environment in place at the
time in question.\(^2\) In the case of a prior point that is on the reasoning
path that lead to the current situation, some sort of standard backtracking
would be sufficient to restore the prior environment in order to examine
the (restored) conflict set involved at that time. In fact, some production
systems already maintain this information for subsequent browsing. In
some of the counter-factual sorts of situations we are interested in,
however, the conflict set of interest is one that would have resulted from
a computational environment that in fact never occurred. We believe that
it will be possible to address this sort of problem using a combination of a
rule-undoing mechanism similar to those sometimes used for
backtracking and a partitioned network or "context" mechanism to isolate
the hypotheticals from the current state of the knowledge-base.

4.3 Intentional Structures

Many aspects of complex reasoning, especially phenomena involving
communication with or understanding of another agent, and/or the
coordination of multiple tasks, ultimately require explicit representation
of structures such as intentions, goals, beliefs and plans. They may also
require the explicit representation of meta-structures that represent the
conflicts and confluences between such mental structures and the actions
that they lead to. [see, e.g., Cohen, et al., 1990, Wilensky 1980] The
approach taken here begins not with such mental state representations,
but rather with an analysis of the forms of rules and their partial matches
into the current state of a knowledge base. It is interesting to note
however, that this syntactic analysis approach can in principle be applied
to counterfactual reasoning involving mental state objects. In
determining that a rule did not fire and thereby assert a particular
conclusion because no object matched one of its variables, or because
there were not appropriate matches to its variables such that the
constraints between the variables could be simultaneously met, there is
nothing said about the nature of the objects the variables are matching.
In particular, such variables can match intentional objects such as goals
just as readily as they can match domain objects, and the explanations
provided should be just as intelligible. We believe that the sort of

\(^2\) Strictly speaking this is true of backward browsing of any kind, but in practise the
browsing back into why facts or constraints did not support a rule are more often
done in a monotonic manner. That is, it is usually ok to simply ask why isn't a fact
true or a rule eligible in the latestest environment, rather than at the time in
question. When the facts were true, and the rule was eligible, but was blocked by
conflict resolution, it becomes impossible to ignore the effects of prior contexts.
counterfactual reasoning we are exploring can be overlaid over any underlying theory of meta-reasoning or planning, since the mental objects of such a theory would simply be the objects to which the rule-base of such a theory would refer.

REFERENCES


