Extracting Knowledge from Expert Systems

John McDermott
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213

To appear in *IJCAI-83*

Abstract. Our understanding of how to use large amounts of knowledge to enhance the problem solving capabilities of computer programs is quite limited. Over the past several years a number of knowledge-based systems have been developed, and this experience has provided us with a handfull of techniques we can apply in a few domains. What we don't have yet is much of an appreciation of why these techniques work or of the limits of their usefulness. In order to take this next step, we need more data. Unfortunately, the analyses of the expert systems currently being built tend to ignore questions that could provide precisely the data needed. This paper proposes a few questions that it might be worth asking and shows how answers to those questions could begin to give us the understanding we lack.

1. Introduction

If those of us involved in expert systems research have a common goal, surely it is to understand how to use large amounts of knowledge in solving problems. We have made some progress. In the 1970's progress promised to be rapid; a small number of interesting systems were developed, and their developers described the tasks the systems could perform and the various techniques used. Over the past few years, many more people have become involved, and expert systems are now springing up all over. Unfortunately, this dramatic growth in the number of expert systems has not, as far as I can see, been accompanied by even a modest growth in our understanding of the relationship between knowledge and search. The problem is a lack of data. Though the number of expert systems that have been developed is now sufficient to allow us to begin to deepen our understanding, the information available about those systems is inadequate.

I am aware of only a few attempts to draw general lessons from an examination of a number of expert systems. A paper by Feigenbaum focuses on several of the systems developed at Stanford's Heuristic Programming Project [Feigenbaum 77]. Although the paper gives some indication of the nature of the task each system addresses, its primary purpose is to provide an existence proof that the by now well known tools of the knowledge engineer can be used with success in a variety of domains. More recently, a group of experienced expert system builders wrote a paper which offers a map linking task characteristics with techniques for solving the task [Stefik 82]; the paper is one of the chapters in a forthcoming book on expert systems [Hayes-Roth 83]. While the paper will be a valuable guide to people trying to construct AI application systems, several of the systems discussed are by no stretch of the imagination knowledge-based; thus the relationship between some of the prescribed

techniques and knowledge-intensive tasks is not at all clear. Two other works containing the insights of experienced expert systems builders will soon be available [Buchanan 83a], [Buchanan 83b]. Though both works provide a number of important insights, they focus almost exclusively on a single task domain. Neither indicates very clearly the scope of the various lessons learned.

This paper assumes that what distinguishes expert systems from other AI systems (and hence what makes them interesting) is that they bring relatively large amounts of knowledge to bear in solving problems. The claim made here is that a few basic aspects of task domains are the sources of the significant variety one finds among expert systems. If systems are described in terms of these aspects, we will gain an understanding of the various roles knowledge can play and an appreciation of the factors that define these roles. In the next section, four aspects of task domains are identified, and then in the third section, three expert systems are described in terms of those aspects. On the basis of these descriptions, several hypotheses about knowledge and its roles in problem solving are generated. The four aspects identified are supposed to be taken seriously; though the set is undoubtedly incomplete, we cannot understand the roles knowledge plays in problem solving without attending to those four aspects. The hypotheses are not to be taken seriously; their function is to suggest what we might learn about knowledge if we had a large number of systems to serve as data points. Since each hypothesis was generated on the basis of at most three data points, all any of them can do is tentatively point the way.

2. Figuring out the questions

In order to understand the roles knowledge plays in any particular task domain, it is necessary to know at least the following sorts of things about the task:

- Compartmentalizability of the task knowledge.
- Uncertainty of the task information.
- Applicability of the task knowledge.
- Thickness of the task knowledge.

How and to what extent task knowledge can be compartmentalized depends almost exclusively on the structure of the task; the more compartmentalized the knowledge, the narrower its scope and the simpler the choice of what pieces of knowledge to apply. Knowledge operates on information made available by the task environment; the more uncertain that information, the less control the knowledge has. Each piece of knowledge has limited applicability; a piece of knowledge is more or less powerful depending on how precisely it specifies the set of situations in which it is applicable. Finally, many different pieces of knowledge may be relevant in the same situation; the thicker the task knowledge, the more knowledge there is to bring to bear at any given time. The rest of this section further elaborates each of these aspects.

2.1. The compartmentalizability of knowledge

Before much sense can be made of compartmentalizability, it is necessary to find a structure that can serve as a compartment. One plausible structure is the subtask. When human experts talk about what they do, they frequently first locate themselves by naming some subtask and then indicate what conditions have to be satisfied in order for some behavior to be appropriate. The subtask name appears to stand for a set of conditions all of which must be satisfied before any of the behaviors associated with that subtask are appropriate. Typically subtasks are organized hierarchically; higher level subtasks are partitioned into a set of smaller, more focused subtasks. It is not clear what the criteria for being a primitive subtask are; presumeably experts differ somewhat with respect to the level to which they decompose the task. But since experts can communicate successfully with others using subtask names, it is unlikely that the decompositions differ to a very great extent. Pieces of knowledge with the same common set of applicability conditions are distinguished from other pieces of knowledge associated with other subtasks. But within the subtask there are no such obvious boundries; in that sense, all of the knowledge associated with the subtask is potentially relevant whenever the subtask is being performed.

Human experts are not at all good at estimating how much knowledge is associated with each subtask. And in fact there is no apparent way of getting a reasonable estimate that does not involve building a system which performs the task. Without an estimate of the amount of knowledge associated with each subtask, it is impossible to understand the access requirements the task imposes. If there is very little knowledge associated with each subtask, it may be possible to statically impose a structure on that knowledge. The more knowledge that is relevant to a subtask, the more likely it is that there will be issues of how to determine dynamically when to apply which pieces of knowledge.

2.2. The uncertainty of information

Two ways in which the task environment strongly shapes the nature of the task are with (1) the reliability of the data it provides, and (2) the points in time, relative to the task, at which the data become available. The unreliability of data is a more or less serious issue depending on a variety of factors. If the agent can assume the data are correct (whether or not they are), then true reliability is irrelevant. If the task environment provides a measure of reliability for each datum, then (unless the measure is itself unreliable) reliability is only a minor issue. If there is ambiguity about the reliability of at least some of the data, but there are ways of determining whether a datum is incorrect within the time frame imposed by the task, then although data reliability is a serious issue, it is at least more tractable than the case in which unreliability cannot be recognized.

If the task environment provides additional data to the agent during the course of the task, it makes a difference who initiates whatever interaction there might be. If the task environment is passive and interactions are initiated only by the agent, the most significant question is what information does the agent have access to (eg, can the agent ask for new information and clarification or confirmation of data already provided?). If the task environment is active and intitiates interactions with the agent, the issue is whether new data can conflict with data provided earlier; if so, a significant question is whether the conflicting data are generated independently of what is happening in the task or because of the effects of the task on the task environment.

2.3. The applicability of knowledge

In any given situation, an agent may have some pieces of knowledge it is certain are applicable, other pieces of knowledge it suspects may be applicable, and presumeably a great deal of knowledge it is certain is not applicable. I will use the term "applicability factor" to distinguish among these three cases. If the conditions on the applicability of an action have an applicability factor of 1, performing the action will result in a transition to a state that is on a solution path. If the conditions of applicability have an applicability factor less than 1 but greater than 0, there is some likelihood that performing the action will result in a transition to a state on a solution path. But an applicability factor less than 1 signals that if the agent performs the action, it may have to backtrack.

There are three reasons why an agent might misuse its knowledge: (1) the agent does not have sufficient information about the current situation (ie, the conditions of applicability on an action are not satisfied), (2) it has misleading information about the current situation, or (3) it does not know the precise conditions under which its knowledge is relevant (ie, the applicability factor is less than 1). In the first case, if the agent can recognize when it is lacking such information and can get the existing information from its external environment, it will not fail to apply relevant knowledge. In the third case, it may be that the agent is simply ignorant; if the conditions under which the knowledge is relevant can be more precisely defined, then the conditions of applicability may be able to be elaborated to the point that the applicability factor becomes 1; see [Doyle 83] for a more thorough discussion of this point. However, if the conditions under which a piece of knowledge is relevant are highly complex or if no one knows them, the possibility of misapplying the knowledge will remain. In such cases it may be possible to extract from an expert or otherwise acquire a reasonably accurate measure of the likelihood that applying the knowledge will result in a state on the solution path; if so, the applicability factor can be taken into account in guiding the search.

It is important to distinguish between tasks in which the agent can know when it has accomplished the task (achieved its goal) and tasks in which this is not possible. In tasks in which the agent uses only knowledge whose applicability factor is 1, it knows when it has accomplished its task because it sees there is nothing more to do. In tasks where some or all of the knowledge used has an applicability factor less than 1, if the agent knows enough about its goal to determine whether it is on a path leading to that goal, then although it may have to search extensively, it will recognize when it has achieved the goal; in such cases, the only role for applicability factors is to make search more efficient. In tasks in which the agent cannot know whether it has achieved its goal, the applicability factors not only can help direct the search, but can also serve as indirect measures of the likelihood that the goal has been achieved.

2.4. The thickness of knowledge

Task knowledge is thick if the same situations can evoke a wide variety of actions depending on the agent's meta-goals. If, for example, the agent can deal with its task at a number of levels of abstraction, its knowledge is thicker than that of an agent who can deal with the task at fewer levels of abstraction. A more interesting example of how the same situation can invoke different knowledge arises when we ask how the association between an action and the conditions of applicability on that action are justified. If someone who claims some piece of knowledge K is always relevant in situation S is asked to justify that claim, the knowledge K' that is used as justification is also relevant in situation S; K' has the same function as K, but presumeably is more basic and thus relevant in many

situations besides S. At the limit is analogy. If problems that arise while a task is being performed can be solved with knowledge having no direct connection to the task, endless opportunities for creative problem solving open up.

3. Converting expert systems to data points

In this section, the aspects of task domains identified above are used to analyze three knowledge-based systems; the results of the analyses are several hypotheses about the effects on problem solving competence of large amounts of domain knowledge. Since only three systems are considered, the analyses will collectively prove to be woefully inadequate, and the hypotheses will be at best suggestive. It will hopefully be clear, however, that if the set of aspects were extended a bit and if fifty or a hundred data points were used instead of three, a number of interesting hypotheses could be generated.

The three systems we will consider were (or are being) developed at CMU for Digital Equipment Corporation. R1 is a computer system configurer [McDermott 82a]; given a set of components, it adds any support components that might be missing and determines what the spatial relationship among all of the components should be. XSEL is a salesperson's assistant [McDermott 82b]; it helps the salesperson select a set of components tailored to the customer's needs and design a floor layout for the resulting system. The XSEL user can call on R1 to configure or reconfigure the current set of component selections and can provide information to R1 which insures that the resulting configuration will satisfy special requirements of individual customers [McDermott 81]. PTRANS is a manufacturing management assistant [Haley 83]; it helps insure that the inevitable problems which arise and threaten to delay the delivery of systems to customers are resolved satisfactorily. In time, XSEL will interact with PTRANS so that a delivery date can be confirmed as soon as an order is booked.

R1's original task was to generate the configuration diagram and do the order-completeness checking required before a VAX-11 order could be built in one of Digital's final assembly and test plants; the original version of R1 was tested in the Fall of 1979 and began to be used on a regular basis in January, 1980, to configure all VAX-11 orders. The information available to R1 consisted of just the set of components (line-items) ordered by the customer; no information was available about how the customer intended to use various components or how he intended to lay them out on the floor. As XSEL was being developed, it became clear that it could collect and pass on to R1 information that would enable R1 to tailor its configurations to the requirements of individual customers; consequently, R1 was extended to be able to use this additional information. More recently R1's knowledge has been augmented, by developers at Digital, so that it can now configure systems other than the VAX-11. The version of R1 discussed below is the version currently used by Digital.

XSEL's development is occurring in two stages. The initial version of the system assists salespeople with both component selection and floor layout design; the assistance provided in component selection assumes the user already has a good appreciation of the customer's computing needs. This version of XSEL began to be field tested in May, 1983. Though its capabilities have been strongly shaped by a user-design group which was formed early in its development, no significant redefinition of XSEL's task has occurred. A major extension to XSEL is being developed at CMU; the extended

system will be an expert in sizing a customer's computing needs. The version of XSEL discussed in this paper is the version which includes the yet to be refined sizing capability.

PTRANS will ultimately be able to assist with the management tasks that arise from the time an order is received by Digital's manufacturing organization until it is delivered to the customer. The initial version of PTRANS, however, deals only with management issues which arise in final assembly and test plants. These plants build complex systems from components produced by high-volume plants. The principal management tasks are determining when and where on the floor to build each system, insuring that the necessary parts are on hand when it is time to issue them to the floor, and tracking the progress of each system on the floor so that problems can be resolved as they arise. In order to perform these tasks, PTRANS must be able to construct plans and then modify these plans appropriately as unforeseen circumstances arise. The initial version of PTRANS began to be tested at Digital in July, 1983, and is the version discussed in this paper.

All three of these systems are implemented in OPS5 [Forgy 81]. OPS5 is a general-purpose rule-based language; like other rule-based languages, OPS5 provides a rule memory, a global working memory, and an interpreter which tests the rules to determine which ones are satisfied by the descriptions in working memory. An OPS5 rule is an IF-THEN statement consisting of a set of patterns which can be matched by the descriptions in working memory and a set of actions which modify working memory when the rule is applied. On each cycle, the interpreter selects a satisfied rule and applies it. Since applying a rule results in changes to working memory, different subsets of rules are satisfied on successive cycles. OPS5 does not impose any organization on rule memory; all rules are evaluated on every cycle.

3.1. The compartmentalizability of knowledge

R1's 280 subtasks conform quite closely to the 200 - 300 subtasks human configuration experts identify. R1 has about 2400 rules distributed among these 280 subtasks. On the average, then, there are 9 rules (pieces of domain knowledge) associated with each subtask; though there is some variation in the amount of knowledge associated with each subtask, for most of the subtasks there are between 5 and 15 relevant rules, and no subtask has more than 30 rules associated with it. Typically 2 or 3 of the rules associated with each subtask recognize when other subtasks must be performed and enable those subtasks by depositing the subtask names in working memory. Almost none of R1's knowledge is relevant to more than one subtask.

XSEL has about 3000 rules distributed among 289 subtasks. All but about 30 of its rules are relevant to only a single subtask; thus, on the average, there are about 10 rules associated with each subtask. At any given time, typically 2 of the 30 rules relevant to more than one subtask are potentially relevant. Thus the total number of rules potentially relevant at any given time is, on the average, about 12.

The tasks PTRANS assists with are performed simultaneously by a number of different people. The responsibility of some of these agents is to insure that the information other agents use is as accurate as possible. Since these monitoring tasks are not associated exclusively with any particular part of the process, their relevance comes and goes as the various tasks which rely on the monitoring tasks come and go. PTRANS has about 1400 rules. Approximately 700 of these are distributed among 175 subtasks; thus there are about 4 rules associated with each subtask. At any given time, 34 of the 700

rules relevant to more than one subtask are potentially relevant. Thus the total number of rules potentially relevant at any given time is, on the average, about 38. The 34 demon rules associated with each task are not easily compartmentalized. The mean number of demons that any two subtasks have in common is 4.3 and the mode is 2; a few subtasks have no demons in common and a few have more than 20 in common. Thus whatever structure might characterize the interrelationships among the demons, it appears to have nothing to do with the task structure. We have had little success so far in identifying any organizational principles at all.

One might expect in tasks like the configuration task, which have a fairly well-defined structure, that only a relatively small amount of knowledge would be potentially relevant at any given time. But the data for all three systems appears to support the following hypothesis:

H1: Only a relatively small amount of an expert's knowledge is potentially relevant in any given situation.

The intriguing question is to what extent this is true of more general problem solvers. It would seem plausible that in domains in which there is not a clear focus on a small set of issues (ie, in which there are not strong expectations about what set of events might occur), much more knowledge would be potentially relevant. What is not at all clear is how much more and whether there is anything at all in common between the sort of compartmentalization of knowledge that we find in expert systems and that which we could expect to find in general problem solvers.

Since the only system with a significant amount of knowledge not associated with specific subtasks is PTRANS and since the knowledge it has that is not associated with specific subtasks appears to be structureless, a possible hypothesis is:

H2: Subtasks serve as the dominant organizing principle for an expert's knowledge.

Much of the knowledge that an expert has appears to be relevant in the context of only a single subtask. Because subtasks often form a hierarchy, this is not quite as restrictive as it may sound; knowledge that is relevant in several subtasks will be associated with a higher level, subsuming subtask. When a piece of knowledge is useful in a variety of subtasks that have no common ancestor, there is no obvious way to characterize the relationship among those uses.

3.2. The uncertainty of information

Because the configuration task as originally defined for R1 did not include direct interaction with a salesperson or customer, the only data available to R1 was that provided at the beginning of the task. This input consists of a list of quantity/component-name pairs and sometimes other information describing customer-specific configuration requirements. The major uncertainty associated with the data is whether the set of specified components are orderable, play together, and are complete; and these uncertainties are precisely those which the task is there to resolve. Any changes to the order, or additional information which a salesperson or customer might want to provide, occur after the task has been performed; such changes define a new configuration task, rather than reflecting uncertainty in the data for the initial task.

Whether XSEL is providing assistance with component selection or with floor layout, it asks the user

for some initial information. In the case of component selection, the information consists (possibly of a mixture) of direct and indirect measures of the customer's computing needs together with an indication of how much the customer is willing to pay to satisfy those needs. In the case of floor layout, the information consists of descriptions of the rooms which will house the system and possibly customer preferences for the placement of some or all of the components. In both cases, there are two ways in which the information provided is unreliable. (1) Viewed as a measurement, a piece of information can simply be incorrect; more frequently the measurement cannot be (easily) made with precision and thus the value provided is only an approximation. (2) Viewed as a constraint, a piece of information can be more or less significant; if the constraints that define the customer's needs cannot all be simultaneously satisfied, some of those constraints must be relaxed. It is sometimes important for XSEL to deal explicitly with these uncertainties. When XSEL provides assistance in sizing a customer's needs, for example, it keeps track of the expected precision of each value; thus when it encounters an inconsistency, it has information it can use to determine which constraints to relax and how much to relax them.

Whenever PTRANS is asked to assist with the processing of an order, it is provided a standard set of initial information. Part of the information is R1's output along with other customer-related information such as the desired delivery date. The other information PTRANS is provided is for the most part related to resource availability (materials, floor space, etc). All of this information is unreliable. If any of the customer-related information changes, a change order is issued; the change order indicates what information has changed and what the new values are. Every resource has time information associated with it that indicates how much of the resource is expected to be available at specific points in the future; whenever an expectation changes, this information is updated. PTRANS is designed to manage (create, implement, and modify) plans precisely because of these kinds of uncertainties; that is, it is built on the premise that these uncertainties are inevitable and thus tries to cushion their impact rather than eliminate them. Because it is impossible to predict the precise nature of the changes (customer and resource) that will occur and exactly where they will occur and because the number of possible changes is extremely large, PTRANS does not explicitly keep track of the uncertainties, but just adapts to the changes when they occur.

Since R1, XSEL, and PTRANS all, though in different ways, directly address issues of data validity, a possible hypothesis is:

H3: An expert's task is almost always, at least in part, one of data validation.

Since R1's task is to determine the completeness and configurability of a set of components, it is primarily a data validation task. XSEL's task often requires drawing the attention of the user to an inconsistency between the resources required and the desired purchase price. PTRANS task is primarily one of responding opportunistically to more current, and thus presumeably more accurate, task information. In all three cases, the reason for the data validation is to simplify or avoid what would otherwise be impossibly complex problems.

Because the initial version of R1 had no way of interacting with salespeople, when it discovered a problem with an order, it had no way of getting the customer-specific information it needed to resolve the problem; thus a possible hypothesis is:

H4: Tasks which are defined in such a way that the environment cannot intrude after the task has begun should always be redefined to allow intrusion.

Until R1's task was redefined, its only recourse when it encountered either a completeness or a configuration problem was to select one of the possible solutions somewhat arbitrarily. In the absence of additional data from the salesperson, it had to make a number of questionable assumptions.

3.3. The applicability of knowledge

Though R1 typically has about 9 rules that are potentially relevant at any given time, ordinarily only 2 or 3 of those rules have conditions that are fully satisfied. That a rule's conditions are satisfied is a necessary but not a sufficient condition for the rule to be applied. In addition a rule must satisfy OPS5's conflict resolution strategies; these strategies order rule instantiations on the basis of considerations such as the recency of the data satisfying the rules and the relative specificity of the rules. Almost all of R1's rules have an applicability factor of 1. That means if a rule's conditions are satisfied and if that rule is not dominated by some other rule on the basis of OPS5's conflict resolution strategies, applying the rule will result in a transition to a state that is on a solution path. Those pieces of R1's knowledge that have an applicability factor less than 1 bear on unibus configuration. R1 is uncertain of the relevance of some of its unibus configuration knowledge because the relevance of the knowledge can only be judged after it is applied. R1 may, for example, begin to create a unibus configuration that turns out to be unacceptable because there is insufficient space; when it recognizes such a situation it backtracks. In order for R1 to have unibus configuration knowledge whose applicability factor is 1, it would have to have knowledge which explicitly identified all valid (or all invalid) unibus configurations.

In many respects the applicability of XSEL's knowledge is very similar in character to R1's. Of the 12 or so rules that are potentially relevant at any given time, ordinarily only 2 or 3 are fully satisfied. Much of XSEL's knowledge has an applicability factor of 1. The knowledge that has an applicability factor less than 1 is that which maps from indirect to more direct measures of computing needs. This knowledge associates some (possibly very indirect) measure of a computing resource need with some set of cues. Since XSEL cannot verify the adequacy of its proposed sizings, it uses applicability factors not only to order its search, but to provide an indication to the user of the degree of variation in possibly acceptable (correctly sized) computer systems. XSEL can recognize when some set of constraints appear to be inconsistent. Before asking the user to relax some of the constraints, it checks to see whether there is an interpretation of the constraints which makes them consistent.

How many of PTRANS' 38 potentially relevant rules are actually relevant on any given cycle depends to a great extent on whether any part of the environment is changing in an unexpected way. When the environment is relatively stable, ordinarily only 1 or 2 rules are actually relevant; but when unexpected changes occur, as many as 10 or 20 demon rules may also be relevant. Most of PTRANS' demon rules have an applicability factor of 1. Most of the rules associated with subtasks have an applicability factor less than 1; this uncertainty manifests itself both in the form we saw in R1 and in the form we saw in XSEL. The knowledge PTRANS has that is like R1's unibus configuration knowledge is used to generate floor assignments; both tasks can be viewed as instances of the general bin-packing problem. The knowledge that is like XSEL's sizing knowledge is used to estimate

the amount of time various events are going to take. XSEL and PTRANS must both predict the future performance of entities they neither fully understand nor control. However the tasks differ because what PTRANS predicts are events that will occur while it is performing its task; thus PTRANS can recognize when it has failed and recover by backtracking.

What is common to those subtasks in which R1, XSEL, and PTRANS search is the relatively large number of different actions that can be appropriate; thus a possible hypothesis is:

H5: If the situations in which an expert can find itself evoke a relatively small number of different behaviors, the pieces of knowledge that the expert uses to determine how to act in those situations will have an applicability factor of 1.

There are many different ways in which a large number of situations can be mapped onto a small number of behaviors. One that is frequently useful for expert systems is to treat a large fraction of the situations as an equivalence class and single out a relatively small number of situations as special cases. Thus there are a few situations that are particularly interesting and the rest evoke an "ordinary" response.

Since R1's domain is narrower than either XSEL's or PTRANS' and since most of R1's knowledge has an applicability factor of 1, a possible hypothesis is:

H6: The narrower the domain, the more likely it is that the expert will almost never search.

I suspect that this hypothesis is false. Currently, R1 is quite expert, seldom needs to search, and yet is able to discriminate among only 2500 different types of situations. XSEL, on the other hand, is as yet much less expert, searches much more frequently, but is able to discriminate among 3000 different types of situations. It is possible that XSEL searches more than R1 because its domain is not as narrow. But it is likely, I think, that XSEL searches more because it is less well developed. I suspect that by the time XSEL has 9000 or 10,000 rules it will be as expert at computer system sizing as R1 is at configuration and will do as little search.

3.4. The thickness of knowledge

For the most part, R1 deals with the world at a single level of abstraction. Its knowledge is primarily about configurable components and their possible interrelationships. It does understand that the task it performs has a hierarchical decomposition, and it performs a variety of abstract tasks. But the knowledge it uses to make its way around in those abstract tasks is still knowledge of configurable components. Moreover, its knowledge is single-level; it does not know why its rules (as opposed to its conclusions) are valid. Though many of its rules could be justified on the basis of more general engineering knowledge, a significant number could be fully justified only by appealing to custom.

XSEL also deals with the world at a single level of abstraction; but its level of abstraction is somewhat higher than R1's. Whereas R1 deals with individual configurable components, XSEL deals with sets of identical configurable components. Like R1, XSEL cannot explain why its rules are valid. However it comes closer than R1 to being able to do so. Its sizing rules associate indicators of need with descriptions of computing resources. Since XSEL has to interact with users of widely varying

degrees of sophistication, these indicators vary widely in their degree of generality. Highly unsophisticated users force XSEL back to first principles.

Because PTRANS' world changes so frequently, it deals with the distant future at a high level of abstraction in an effort to avoid getting bogged down in ephemeral details. It deals with the present and near future at a variety of levels of abstraction depending on the nature of its current subtask. To perform its various scheduling functions, it requires very little information about the objects it is dealing with. But to provide assembly and testing assistance to technicians, it must descend into a rococo world similar in many respects to R1's. Like R1 and XSEL, PTRANS cannot explain why its rules are valid.

Since R1 deals with computer systems at a lower level of abstraction than XSEL and since R1 does almost no search, a possible hypothesis is:

H7: The more abstract a piece of knowledge is relative to other knowledge about the same situation, the lower its applicability factor will be.

In order for a piece of knowledge to have an applicability factor of 1, it must be possible to specify precisely the situations in which the knowledge is relevant; that requirement delimits the base level of abstraction for that piece of knowledge. If the knowledge is to be used in a more abstract space, not all of the information required to determine the applicability of the knowledge will be available.

Since R1, XSEL, and PTRANS all have quite thin knowledge and since all of them occassionally encounter problems they cannot resolve and all of them must do at least some search, a possible hypothesis is:

H8: Thick knowledge reduces the number of intractable problems; it can also reduce the amount of search.

When a expert encounters a difficult problem, he can sometimes uncover the knowledge needed to solve the problem by trying to justify the knowledge ordinarily used to solve such problems. Presumeably this strategy works because as different layers of knowledge are built up for different purposes, pieces of knowledge that would be highly redundant at some level are not included at that level; these omissions reduce the size of the search space, but at the cost of incompleteness.

4. Concluding remarks

It occured to me, as I began working on this paper, that it might have more validity if it included, as data points, systems from a wide variety of backgrounds, rather than just systems developed at CMU. As I pursued this idea, however, I discovered that the kind of system data I wanted is not very easy to find in the literature. That may, of course, be because the data I was looking for is not very valuable, or is odd, or is peculiar to a particular approach to developing systems. But I suspect not. The area of knowledge-based systems is new enough that we have not yet had time to define our needs. The purpose of this paper is not to define those needs, but to raise the issue and make some stabs.

One general point that I think does come through even from the narrow base I have provided is that the analyses of expert systems and their tasks is likely to look quite different from that of classical

search-intensive systems and tasks. Knowledge is certainly used in those programs, but it is a very small amount of general knowledge, mostly with a low applicability factor.

Given just the data available from R1, XSEL, and PTRANS, the hypotheses presented above are not implausible. But mostly they are simplistic. What is surely missing is an adequate amount of data from which to generate substantive hypotheses. Though my shadow hypotheses lack substance, they do suggest the sort of advantages substantive hypotheses could confer. First, having a partial description of a role that knowledge can play gives us something to refine and exploit or falsify; we can focus our efforts on pushing and shaping promising insights. Second, substantive hypotheses also provide an avenue of discovery; given a set of hypotheses to reflect on, we are in a position to discover holes (approaches that have not been tried). But we are not going to have substantive hypotheses about the role of knowledge in problem solving unless our analyses of the expert systems we build lay bare the natures of their tasks.

Acknowledgements

The ideas in this paper have evolved as a result of discussions with many people. Helpful comments on earlier drafts were made by Danny Bobrow, Bruce Buchanan, Dick Duda, Bob Engelmore, Lee Erman, Paul Haley, Gary Kahn, Dave McKeown, Allen Newell, Dennis O'Connor, Ted Shortliffe, and Mark Stefik; I am sorry to say that in several cases, I was not able to do justice to their suggestions. Paul Haley and Kemal Oflazer helped with some of the data analysis.

References

[Buchanan 83a] Buchanan, B. and R. Duda.
Principles of rule-based expert systems.
In Yovits, M. (editor), *Advances in Computers 22*, . Academic Press, forthcoming, 1983.

[Buchanan 83b] Buchanan, B. and E. Shortliffe.
Expert Systems Research: The Mycin Experiments of the Stanford Heuristic Programming Project.
Addison-Wesley, forthcoming, 1983.

[Doyle 83] Doyle, J.
Methodological simplicity in expert system construction.
Technical Report, Carnegie-Mellon University, Department of Computer Science, 1983.

[Feigenbaum 77] Feigenbaum, E. A.
The art of artificial intelligence: themes and case studies in knowledge engineering.
In *Proceedings of the 5th International Joint Conference on Artificial Intelligence.* Boston, 1977.

[Forgy 81] Forgy, C. L.
OPS5 User's Manual.
Technical Report, Carnegie-Mellon University, Department of Computer Science, 1981.

[Haley 83] Haley, P., J. Kowalski, J. McDermott, R. McWhorter.
PTRANS: A rule-based management assistant.
Technical Report, Carnegie-Mellon University, Department of Computer Science, in preparation, 1983.

[Hayes-Roth 83] Hayes-Roth, R., D. Waterman, and D. Lenat.
Building Expert Systems.
Addison-Wesley, forthcoming, 1983.

[McDermott 81] McDermott, J. and B. Steele.
Extending a knowledge-based system to deal with ad hoc constraints.
In *Proceedings of the 7th International Joint Conference on Artificial Intelligence.* Vancouver, 1981.

[McDermott 82a] McDermott, J.
R1: A rule-based configurer of computer systems.
Artificial Intelligence 19(1), September, 1982.

[McDermott 82b] McDermott, J.
XSEL: A computer sales person's assistant.
In Hayes, J. E., Michie, D., Pao, Y-H (editors), *Machine Intelligence 10*, . John Wiley & Sons, 1982.

[Stefik 82] Stefik, M., J. Aikins, R. Balzer, J. Benoit, L. Birnbaum, F. Hayes-Roth, and E. Sacerdoti.
The organization of expert systems, a tutorial.
Artificial Intelligence 18(2), March, 1982.