SEEK2: A GENERALIZED APPROACH TO AUTOMATIC KNOWLEDGE BASE REFINEMENT

Allen Ginsberg, Sholom Weiss
Department of Computer Science, Rutgers University
and
Peter Politakis
Digital Equipment Corporation

ABSTRACT

This paper describes an approach to *knowledge base refinement*, an important aspect of knowledge acquisition. Knowledge base refinement is characterized by the addition, deletion, and alteration of rule-components in an existing knowledge base, in an attempt to improve an expert system's performance. SEEK2 extends the capabilities of its predecessor rule refinement system, SEEK [1]. In this paper we describe the progress we have made since developing the original SEEK program: (a) SEEK2 works with a more general class of knowledge bases than SEEK, (b) SEEK2 has an "automatic pilot" capability, i.e., it can, if desired, perform *all* of the basic tasks involved in knowledge base refinement without human interaction, (c) a metalanguage for knowledge base refinement has been specified which describes both domain-independent and domain-specific metaknowledge about the refinement process.

I KNOWLEDGE ACQUISITION AND THE KNOWLEDGE BASE REFINEMENT PROBLEM

The problem of constructing an efficient and accurate formal representation of an expert's domain knowledge, the *knowledge acquisition problem*, is a key problem in AI.* As a practical matter, the most difficult aspect of expert system design is the construction of the knowledge base; the rate of progress in developing useful expert systems is directly related to the rate at which it is possible to construct good knowledge bases.

The knowledge acquisition problem can be divided into two phases. In phase one the knowledge engineer extracts an initial rough knowledge base from the expert, "rough" in the sense that the overall level of performance of this knowledge base is usually not comparable to that of the expert. In the second phase, the *knowledge base refinement phase*, the initial knowledge base is progressively refined into a high performance knowledge base. In terms of a rule-based knowledge base, phase one involves the acquisition of entire rules, indeed entire sets of rules, for concluding various hypotheses. The refinement phase, on the other hand, is characterized not so much by the acquisition of entire rules but by the addition, deletion, and alteration of rule-components in certain rules in the existing knowledge base, in an attempt to improve the system's performance. Obviously the foregoing description of knowledge base construction is an idealization. In practice the line between these two phases is not as sharply drawn.**

A knowledge base refinement problem can be thought of as an optimization problem in which we start with a proposed general solution to a given set of domain problems and the goal is to refine it so that a superior solution is obtained. The proposed solution is a working knowledge base that is in need of minor adjustments, but not a major overhaul, i.e., one assumes that the rules given by the expert are basically "sensible" propositions concerning the problem domain. The refinements applied to the rules of this knowledge base must not only meet the obvious requirements of being syntactically and semantically admissible, they must also be conservative, in the sense that they *tend to preserve, as far as possible, the expert's given version of the rules.* Employing rule refinements that meet these requirements makes it more likely that the construction of a refined knowledge base will not simply be a matter of "curve fitting," but will result in a knowledge base that is more robust yet at the same time "close" to the actual knowledge of the expert.

---

*This research was supported in part by the Division of Research Resources, National Institutes of Health, Public Health Service, Department of Health, Education, and Welfare, Grant P41 RR02230.

**In this paper we limit our concern to knowledge bases that are structured collections of production-rules. Unless otherwise stated, when we use the term "expert system," we mean, properly speaking, "production-rule based expert system." Furthermore we confine our attention to refinements of production rules that can be achieved as the result of the sequential application of certain generic refinement operations that are either generalization or specialization operations.

II THE BASIC APPROACH: EMPIRICAL ANALYSIS OF RULE BEHAVIOR USING CASE KNOWLEDGE

A. Overview

In this section we briefly review the basic approach to knowledge base refinement taken by SEEK [1] - an approach that we have continued to employ in SEEK2.

A fundamental assumption of this approach is that *case knowledge* can be used to drive a process involving *empirical analysis of rule behavior* in order to generate plausible suggestions for rule refinement. Case knowledge is given in the form of a data base of cases with known conclusions, i.e., each case contains not only a record of the case observations but also a record of the expert's conclusion for the case. Empirical analysis of rule behavior involves gathering certain statistics concerning rule behavior with respect to the data base of cases; suggestions for rule refinements are generated by the application of refinement heuristics that relate the statistical behavior and structural properties of rules to appropriate classes of rule refinements. We will shortly explicate the nature of the statistical evidence gathered and give an example of these rules.

Our basic control strategy is a combination of "divide and conquer" together with a goal-directed backward chaining mechanism. We assume that the expert and knowledge engineer can identify a finite set of *final diagnostic conclusions* or "endpoints;" these are the conclusions that the expert uses to classify the given cases. One can then confine one's attention to the refinement of rules that are involved in concluding a particular endpoint, e.g., if the domain is Rheumatology one may decide to work on refining those rules involved in concluding the single final diagnosis Systemic Lupus. This is the divide and conquer part of the strategy; it means that at any given moment the system is applying the refinement heuristics only to a proper subset of the rules in the domain knowledge base. The goal-directed backward chaining mechanism comes into play once an endpoint has been chosen. If our chosen endpoint is Systemic Lupus, for example, we begin by applying the heuristics to all the rules in the knowledge base that *directly conclude Systemic Lupus*, i.e., rules whose right hand side is this conclusion. A rule that directly concludes some endpoint will, in general, have components on its left hand side that themselves are the conclusions of some other rules; such components are called *intermediate hypotheses*. The rules that conclude intermediate hypotheses may themselves include components that are intermediate hypotheses. Whenever the refinement heuristics suggest modifying an intermediate hypothesis IH, such as deleting it from some rule, the rules that conclude IH are thereby implicated as candidates for refinement.

B. Some Statistics and A Heuristic

At the highest level, many refinements of production rules may be thought of as falling in one of two possible classes: *generalizations and specializations*. By a rule generalization we mean any modification to a rule that makes it "easier" for the rule's conclusion to be accepted in any given case. A generalization refinement is usually accomplished by deleting or altering a component on the left hand side of the rule or by raising the *confidence factor* associated with the rule's conclusion. By a rule specialization we mean modifications to a rule that make it "harder" for the rule's conclusion to be accepted in any given case. A rule specialization is usually accomplished by adding or altering a component on the left hand side or by lowering the confidence factor associated with the rule's conclusion [2].

On the side of evidence for rule generalization, one of the concepts we have employed in both SEEK and SEEK2 is a statistical property of a rule computed by a function that we call *Gen(rule)*. Gen(rule) is the number of cases in which (a) this rule's conclusion *should have been reached but wasn't*, (b) had this rule been satisfied the conclusion would have been reached, and (c) of all the rules for which the preceding clauses hold in the case, this one is the "closest to being satisfied." A measure of how close a rule is to being satisfied in a case, based on the number of additional findings required for the rule to fire, is easily computed given the case data (for details of the algorithm used by SEEK see [3]; SEEK2's closeness measure is essentially the same).

On the side of evidence for rule specialization, one of the concepts we have defined is a statistical property of a rule that is computed by a function we call *SpecA(rule)*. SpecA(rule) is the number of cases in which (a) this rule's conclusion *should not have been reached but was*, and (b) if this rule had *failed to fire* the correct conclusion would have been reached, i.e., the correct conclusion was the "second choice" in the case (due to its having the second highest confidence), and the only circumstance preventing its being the "first choice" is the fact that this rule is satisfied. If there is *more than one satisfied rule* that concludes the incorrect first choice then none of these rules has its SpecA measure incremented; instead we have defined an additional concept to cover this situation called *SpecB(rule)*: each of these rules has its SpecB measure incremented.

To get a feeling for the sort of heuristics employed by these systems, suppose that for a certain rule r it has been found that Gen(r) > [SpecA(r) + SpecB(r)], in other words the evidence suggests that it is more appropriate to generalize than specialize r. Another piece of information would help us decide *which component of r* should be deleted or altered, viz., the *most frequently missing component*, i.e., the component of r that has the lowest frequency of satisfaction relative to the cases that contribute to Gen(r).

The function that computes this statistic is called *Mfmc(rule)*. Mfmc(rule) also tells us the *syntactic category* of this most frequently missing component. For example, one sort of component often used in medical diagnostic systems is called a *choice component*. These have the form $[k: C_1, ..., C_n]$, where *k, the choice number* is a positive integer and the C_i's are components (findings or hypotheses, but not choices). A choice component is satisfied *iff* at least *k* of its C_i's are satisfied. If we know that the rule r should be generalized and that Mfmc(r) is a particular choice component, then a natural thing to do is to *decrease the choice number of that choice component*. Being conservative we decrease the choice number by 1.

To summarize the discussion in this section we now display in full the particular heuristic we have described.

If: `Gen(rule) > [SpecA(rule) + SpecB(rule)] & Mfmc(rule) is CHOICE-COMPONENT C`

Then: `Decrease the choice-number of CHOICE-COMPONENT C in rule.`

Reason: `This would generalize the rule so that it will be easier to satisfy.`

III THE SEEK EXPERIENCE

A salient feature of the original SEEK program [1] is that it was not designed to solve the entire knowledge base refinement problem on its own, rather it was intended to help, interactively, an expert or knowledge engineer solve the overall problem by offering potential solutions to various sub-problems that arise along the way. SEEK helps its user in the following ways: (a) it provides a performance evaluation of the knowledge base relative to the case data base, (b) using its statistical concepts and heuristics it identifies rules that are plausible candidates for refinement and suggests appropriate refinements, (c) a user can instruct SEEK to calculate what the actual performance results of a particular refinement to the knowledge base would be, and if the user desires, SEEK will incorporate the change in the knowledge base.

A. Basic Cycle of Operation

Although control in SEEK always resides with the user, and there are a number of paths and facilities available to the user at almost every point, SEEK can be thought of as having a basic cycle of operation. The system is given an initial knowledge base and the case knowledge data base. SEEK first obtains a performance evaluation of the initial knowledge base on the data base of cases. This is done by "running" the initial knowledge base on each of the cases in the data base, and then comparing the knowledge base's conclusion with the stored expert's conclusion. The performance evaluation consists primarily of an overall score, e.g. 75% of cases diagnosed correctly, as well as a breakdown by final diagnostic category of the number of cases in which the system agrees with the expert in reaching a particular diagnosis, i.e., "true positives," and the number of cases in which the system reaches that diagnosis but the expert does not, i.e., "false positives."

The user must decide on a diagnosis for which he would like to see refinements in the knowledge base in order to obtain better performance, e.g., if the domain is Rheumatology the user may decide to try to upgrade the system's performance in diagnosing Systemic Lupus. For the sake of brevity, we call this user-specified diagnosis the GDX for the current cycle of operation, where the "G" stands for "given," since this is a directive that the user must give the system. The next part of the cycle involves computing statistical properties concerning the rules of the knowledge base that conclude the GDX. Plausible refinements are then generated by evaluating a set of heuristics similar to the one presented above for each of these rules, as well as any rule that becomes implicated via an intermediate hypothesis (see section II.A above).

Once SEEK has given its advice - we think of each piece of advice as a possible "experiment" to improve the knowledge base - the user will initiate an *experimentation* phase. This is a sub-cycle in which the user, interacting with SEEK, determines the exact effect of incorporating any one of the proposed experiments. The user will then decide which, if any, of these refinements should be accepted, and instructs SEEK accordingly. This ends the basic cycle, which can now be repeated starting with the modified knowledge base. This process continues until the user is satisfied with the overall performance evaluation.

B. Limitations

One of SEEK's limitations has already been mentioned: it does not have the capability to attempt to solve a refinement problem on its own. We discuss how SEEK2 removes this limitation in section IV.A below.

Another important limitation of SEEK is that it does not work with a general production rule system, rather it expects that the domain knowledge base will be written in a form known as the *criteria table representation*. This mode of representation requires the knowledge engineer to specify a list of "Major" observations and a list of "Minor" observations for each possible (diagnostic) conclusion in the knowledge base. Rules for reaching particular conclusions are then stated in terms of the number of Majors and Minors for the conclusion, "Requirements" and "Exclusions." The latter are additional observations or conclusions, or conjunctions of such, that are relevant to the diagnosis: a Requirement is some condition that must be satisfied to reach the conclusion; an Exclusion is some condition that "rules out" the conclusion. Furthermore, any rule can reach its conclusion at one of only

three possible confidence levels, viz. possible, probable, definite. As an example, assuming that a list of majors and minors for the conclusion Systemic Lupus has been specified, a rule for concluding the latter might state that if (at least) two of the majors and two of the minors are present then the conclusion is warranted at the "definite" level.

While this mode of representation has proven to be useful in the Rheumatology domain [4] and other medical applications, it is in fact not as powerful a representation language as that of EXPERT [5] or similar production rule systems, in the sense that one can write knowledge bases in general production rule languages that are not translatable into the criteria table format. However, any criteria table can be translated into production rule syntax. *Thus the set of criteria table knowledge bases is a proper subset of the set of production-rule knowledge bases.*

SEEK's knowledge engineering knowledge, i.e., its statistics and heuristics, was formulated with reference to the criteria table representation scheme, and criteria table concepts also were embedded in the control structure of the program. As a consequence, certain forms of rule refinement were not available or were restricted in SEEK, e.g., changing a rule's confidence factor was limited to making "jumps" from one level to another, such as probable to possible. In general, SEEK could do very little with a knowledge base that was not written in a criteria table format.

SEEK2, on the other hand, is a refinement system that will work with any knowledge base written in EXPERT's rule representation language. In designing SEEK2, we found it was possible to decouple SEEK's knowledge engineering concepts from the criteria table representation; we were able to apply many of these concepts in relation to features of more general types of production-rules. For example, criteria table rule-components using the notion of Majors and Minors are special cases of rule-components using choice-functions. Decreasing or increasing the number of Majors or Minors required by a rule, is a special case of decreasing or increasing the choice-number of a choice-function. Thus the example heuristic given above in section II.B is a generalization in SEEK2 of two separate similar heuristics originally stated in SEEK, one for Majors, one for Minors.

In moving to a more general representation language as the target language for knowledge base refinement, we broadened the scope of the set of generic refinement operations available to the system. For example, confidence factors for generalization experiments may be increased based on an average of the highest-weighted (erroneous) conclusion for a set of misclassified cases.

From the programmer's point of view, SEEK's *own knowledge base*, the representation of its knowledge engineering statistics and heuristics, was strictly separate from its control structure. However, this was not the case from the point of view of the user, since there was no facility by which the user, *qua user* and not *qua* programmer, could access and modify SEEK's own knowledge base, in the way that a user can modify the domain knowledge base. Our approach to this issue forms part of a broader project which we describe in section IV.B.1 below.

IV SEEK2: AN EXPERT SYSTEM FOR KNOWLEDGE BASE REFINEMENT

A. Automatic Pilot Capability

Unlike SEEK, SEEK2 is a system that can present plausible solutions to the overall refinement problem *without* the need for interaction with an expert. The output of SEEK2 running in automatic pilot mode is not a list of suggested rule refinements for a particular GDX ("given" dx), rather it is a refined version of the entire knowledge base, i.e., a set of rule refinements to the initial knowledge base which yield an improvement in overall performance. In this section we describe SEEK2's current Automatic Pilot capability.

The attempt to find a sequence of refinements that optimizes performance is a search problem. Where there is a search problem of sufficient complexity, good heuristics must be found to guide the search. As we will see, SEEK2's current automatic pilot algorithm is a heuristic search algorithm, in the sense that it uses a classic "weak method," hill-climbing.

When running in automatic pilot mode SEEK2 makes three types of decisions that were previously made by the user of SEEK: (a) choice of GDX for the current cycle, (b) which rule refinement experiments to try, (c) which refinements to incorporate in the knowledge base given the results of the experiments (see figure IV-1). Additionally SEEK2 has to know when to stop.

In the current implementation, SEEK2 orders the potential GDX's in descending order according to a simple measure on the number of "false negatives" and "false positives," information that is given by the performance evaluation phase. Potential rule refinement experiments for a GDX are ordered by simple measures on the statistics used in generating the refinement, e.g., if the generalization heuristic given in section II.B fires, the quantity Gen(rule) - [SpecA(rule) + SpecB(rule)] is used as an estimate of the *expected net gain* to be derived by performing the experiment.

Information of this sort *could* be used to limit the number of experiments performed in a cycle. However, in the current implementation, the information is used only to determine the order in which GDX's are chosen and experiments attempted; ultimately *every* potential GDX (for which perfect performance has not been obtained) is chosen, and every experiment suggested by the heuristics is performed.

In other words, an automatic pilot cycle involves attempting, according to the ordering just given, every proposed refinement experiment for every potential final diagnostic conclusion in the knowledge base. (Of course, the number of experiments generated by the heuristics represents a small fraction of the total number of logically admissible refinements.) Out of all these experiments SEEK2 "accepts" only one, the one that gives the greatest *net gain* in knowledge base performance for *all* final diagnostic conclusions (*not just* for one GDX). An internal record of the accepted refinement is kept; and then the next automatic pilot cycle begins. If the current automatic pilot cycle is such that no attempted experiment leads to an actual net gain, SEEK2 stops.

We present a simplified example in order to illustrate the preceding remarks. Let us suppose that we have a Rheumatology knowledge base dealing only with the two final

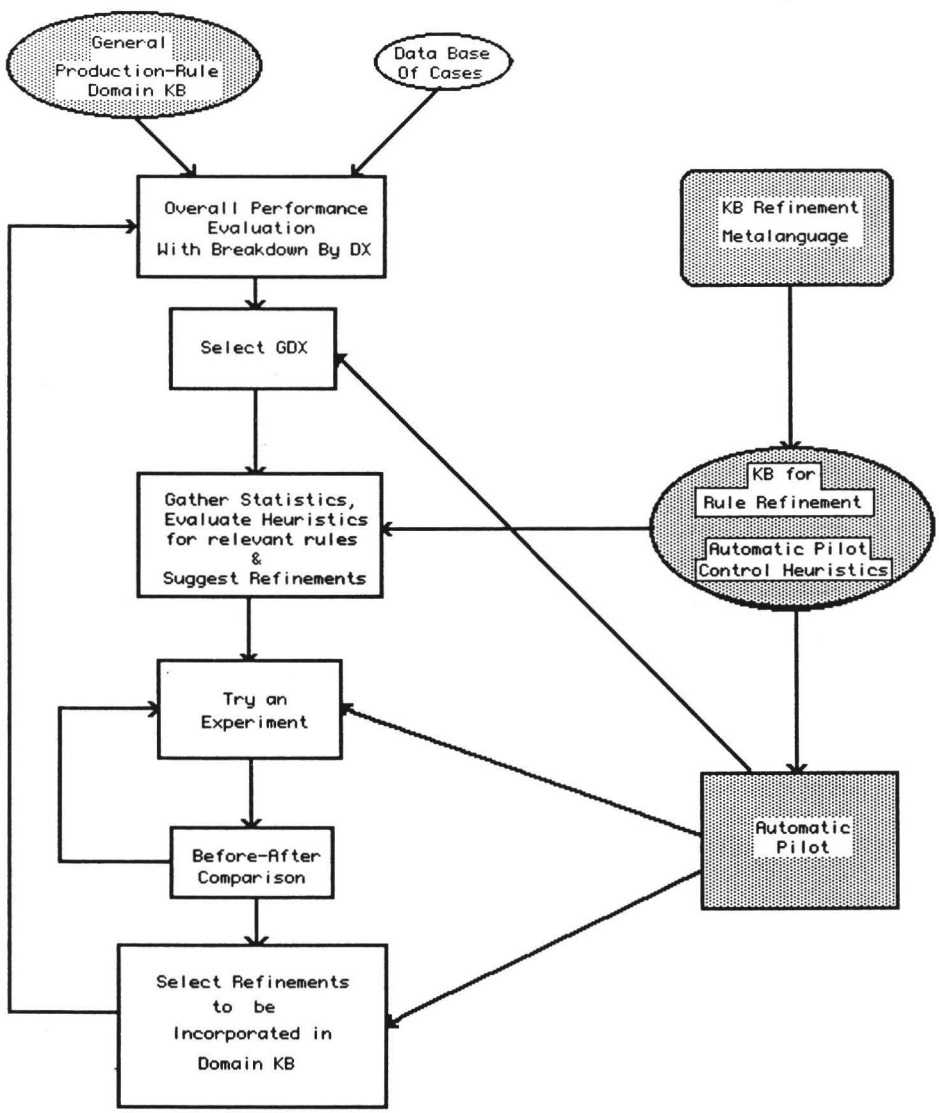

Figure IV-1: Overview of SEEK2; shaded portions indicate extensions to SEEK

diagnoses Systemic Lupus and Rheumatoid Arthritis, that a data base of 20 cases is available, and that our human expert has diagnosed 10 of these cases as Systemic Lupus and the other 10 as Rheumatoid Arthritis. Suppose that the initial performance evaluation computed by SEEK2 is as follows:

DX	True Positives	False Positives
Rheumatoid Arthritis	9 / 10	5
Systemic Lupus	3 / 10	1
Total	12 / 20	

The measure SEEK2 uses to compute GDX order is the maximum of the false negatives and false positives. Thus in our example Systemic Lupus would be the first diagnosis in the GDX ordering since it has 7 false negatives, i.e., 7 out of the 10 cases that should have been diagnosed as Systemic Lupus were not. Therefore SEEK2 will first generate refinement experiments for Systemic Lupus.

Continuing the example, suppose that rule r concludes Systemic Lupus, and SEEK2 finds that Gen(r) = 6, SpecA(r) = 1, SpecB(r) = 0, and Mfmc(r) = Choice-Component C. These findings would satisfy the antecedent of the refinement heuristic presented in section II.B. Therefore SEEK2 will post the decreasing of C's choice-number as a refinement experiment. SEEK2's estimate of the expected net gain of performing this experiment is given by Gen(r) - [SpecA(r) + SpecB(r)] = 5. (This is an estimate; the only way to know what the precise effect of decreasing the choice-number of C will be, is to decrease it, and then recompute the system's performance on the entire data base of cases.) Once all the refinement experiments for Systemic Lupus have been posted and ordered according to their expected net gain, SEEK2 performs all the experiments on this list as ordered. If SEEK2 finds that decreasing the choice-number of component C in rule r leads to an overall performance gain of 3 cases, i.e., the "bottom line" performance total for *both* Rheumatoid Arthritis and Systemic Lupus improves from 12 to 15/20, and this turns out to be the maximum net gain of all the experiments for Lupus, SEEK2 records this fact.

Next, it will select Rheumatoid Arthritis as the GDX, and repeat the process. Suppose that the aforementioned experiment for rule r yields a greater net gain than the best refinement experiment for Rheumatoid Arthritis. Then SEEK2 will "accept" the refinement to rule r, i.e., it will modify its internal copy of the domain knowledge base to reflect this refinement, and a new cycle will commence.

The automatic pilot algorithm just described is a hill-climbing procedure: at each step SEEK2 is guided totally by the "local" information as to which proposed refinement on the current knowledge base results in the best improvement, i.e., leads in the direction of "steepest ascent." When SEEK2 stops it is because a maximum has been reached. This may very well be only a local maximum. While a local maximum represents a "dead end" to the current SEEK2, we are experimenting with special statistics and heuristics that will "kick in" only when a dead end is reached, and which will hopefully allow the system to discover a better maximum if one exists.

B. A Metalanguage for Representing Meta-level Knowledge

1. Metaknowledge in Knowledge Base Refinement

It is clear from the examples in section II.B that a refinement system requires metaknowledge of both the syntax and semantics of the "object" system's language, i.e., the representation language of the domain knowledge base. For example, Gen, Mfmc, SpecA, and SpecB presuppose a working knowledge of what it is for a rule or a rule-component to be satisfied. Other researchers have shown ways in which metaknowledge can aid in the general knowledge acquisition process [6] and in enhancing an expert system's performance [7]. In this section we describe a metalanguage designed specifically for the refinement task. Using this metalanguage one can define knowledge engineering concepts and heuristics, such as *Gen(rule)*, as well as domain specific metaknowledge - e.g., the fact that case findings x and y are incompatible - in terms of a set of primitive concepts and operations.

One motivation for a metalanguage was alluded to in section III.B, where we mentioned that SEEK's knowledge base of heuristics and statistics was inaccessible to the user of the system. The ability to easily access and modify this knowledge base is quite desirable for designing and experimenting with refinement concepts. For example, some of the current statistics for SEEK2 are not likely to be as useful with respect to an expert system that employs a scoring scheme for combining confidence factors. Useful variants of these statistics could be defined within the same metalanguage that we have developed for SEEK2.

In general, even within one expert system framework, different styles of knowledge bases are possible; it is likely therefore that different styles of refinement will be needed. For example, some knowledge bases employ a *taxonomic* ordering of hypotheses. Such an ordering provides knowledge that could be used, together with appropriate control heuristics, to formulate a more efficient version of SEEK2's automatic pilot algorithm. A knowledge base refinement metalanguage will allow for the representation of such control heuristics (see figure IV-1). A refinement metalanguage will allow the expert or knowledge engineer to represent the knowledge that a certain component in a rule should not be altered under any circumstances, or that if a change to component x is made, component y must be changed as well, or that additions to any rule with conclusion C should be drawn from a specified list of components, etc. It will also

allow for the definition of special-purpose statistics that may be of use in suggesting plausible rule refinements, e.g., the frequency of occurrence of a certain combination of findings in a specified subset of cases.

2. Defining Statistics in SEEK2's Metalanguage

SEEK2's statistical concepts can be specified in a set-theoretic metalanguage that employs only a small number of *refinement primitives* together with some appropriate notions from simple set theory, arithmetic, and logic. Using these primitives it is possible to experiment with variations on SEEK2's statistics and define domain specific statistics as well.

A set-theoretic definition of concepts such as *Gen(rule)* (see section II.B) requires refinement primitives of the following sorts. Some primitive variables are needed to provide the system or a user with the ability to "access" various "objects" in the domain knowledge base and the data base of cases. For example, *rule* is a variable whose range is the set of rules in the domain knowledge base, *case* is a variable whose range is the set of cases in the data base of cases, and *dx* is a variable whose range is the set of possible final diagnostic conclusions in the knowledge base. In addition some primitive *functions* are needed to allow one to refer to selected parts or aspects of a rule or a case, e.g., *RuleCF(rule)* is a function whose value is the confidence factor associated with *rule*, *PDX(case)* is a function whose value is the expert's conclusion in *case* ("PDX" stands for "Physician's Diagnosis"), and *CDX(case)* is a function whose value is the conclusion reached for the current knowledge base in *case* ("CDX" stands for "Computer's Diagnosis"). As an example of the way in which these primitives can be used, note that using the notions of PDX(*case*) and CDX(*case*) one may define a *misdiagnosed case* as any *case* for which PDX(*case*) ≠ CDX(*case*).

Certain special sets of objects are of importance in the knowledge base refinement process, and it is therefore useful to have primitives that refer to them, e.g., *Rules-For(dx)* is a function whose value is the set of rules that have *dx* as their conclusion. Finally various primitives that in some way involve *semantic* properties of rules, or the *performance characteristics* of the knowledge base as a whole are useful, e.g., *Satisfied(rule-component, case)* is a predicate that is true *iff rule-component* is satisfied by the findings in *case*, and false otherwise, and *ModelCF(dx, case)* is a function whose value is the system's confidence factor accorded to *dx* in *case*.

SEEK2's refinement knowledge base was designed using the metalanguage we have just outlined [8]. Implementation was achieved by incorporating the aforementioned primitives as procedures and functions, and then coding (by hand) high-level set-theoretic definitions as efficient procedural forms employing these primitives. Currently we are experimenting with a system in which the primitives described above (and others) are available to the user and can be combined to form expressions designating sets, variables, and functions of interest to the user.

V DISCUSSION

SEEK2 currently has ten statistical concepts and nine heuristics for generating refinements. Working in automatic pilot mode on a Rheumatology knowledge base of approximately 140 rules with 5 final diagnostic categories, and using a data base of 121 cases, SEEK2 was able to increase the overall performance of the system from a value of 73% (88/121) to a value of 98% (119/121). It used approximately 18 minutes of Vax-785 cpu time. The total number of experiments tried was 106, out of which 8 were accepted.***

In evaluating the usefulness of SEEK2's automatic pilot capability it is important to keep in mind that the expert is still the final judge. Despite the assured gain in performance with respect to the *given* data base of cases, and the reasonable expectation of performance enhancement with respect to *new* cases, the expert may agree with only a subset of the total number of refinements suggested by SEEK2****. The measure of SEEK2's usefulness is not, however, simply how many of its experiments the expert accepts; even rejected experiments have value: they point out areas of the knowledge base that need to be examined if enhanced performance is to be achieved.

Validity and consistency are important goals in developing expert systems. Yet the design of these systems is often lacking in a coherent formal approach for achieving these goals. The approach to knowledge base refinement described here can lead to a more solid foundation for designing and validating expert system knowledge bases.

VI ACKNOWLEDGMENTS

We would like to thank Casimir Kulikowski for his helpful suggestions concerning this work, and Kevin Kern for programming assistance.

***The system has not yet been tested on alternative knowledge bases.

****The incorporation of domain-specific metaknowledge in SEEK2 might enable the system itself to sometimes reject a refinement that in some way violates the expert's understanding of the domain, even though it may improve performance.

References

[1] Politakis, P. and Weiss, S. "Using Empirical Analysis to Refine Expert System Knowledge Bases." *Artificial Intelligence.* 22 (1984) 23–84.

[2] Mitchell, T. "Generalization as Search." *Artificial Intelligence.* 18 (1982) 203–226.

[3] Politakis, P., *Using Empirical Analysis to Refine Expert System Knowledge Bases,* PhD dissertation, Department of Computer Science, Rutgers University, 1982.

[4] Lindberg, D., Sharp, G., Kingsland, L., Weiss, S., Hayes, S., Ueno, H., and Hazelwood, S. "Computer-Based Rheumatology Consultant." In *Proceedings of the Third World Conference on Medical Informatics.* North-Holland, 1980, 1311–1315.

[5] Weiss, S., and Kulikowski, C. "EXPERT: A System for Developing Consultation Models." In *IJCAI-79.* Tokyo, Japan, 1979, 942–947.

[6] Davis, R. "Interactive Transfer of Expertise: Acquisition of New Inference Rules." *Artificial Intelligence.* 12 (1979) 121–157.

[7] Li-Min, Fu and Buchanan, B. "Enhancing Performance of Expert System by Automated Discovery of Meta-Rules." In *The First Conference on Artificial Intelligence Applications.* December, 1984.

[8] Ginsberg, A., Weiss, S., and Politakis, P. "An Overview of the SEEK2 Project", Technical report, Department of Computer Science, Rutgers University, 1985.