Int. J. Man–Machine Studies (1991) **35**, 789–805

Ameliorating the *Pregnant Man* problem: a verification tool for personal computer based expert systems

JOHN M. HEROD AND A. TERRY BAHILL

Systems and Industrial Engineering, University of Arizona, Tucson, AZ 85721, USA

(Received 26 June 1989 and accepted in revised form 21 June 1990)

With the increase in expert systems development over the last several years, has come a need for verification tools that will allow the knowledge engineer to debug and refine the systems. Although some tools are forthcoming, these tools have traditionally been concerned with completeness and consistency checking of the knowledge base itself. Completeness and consistency checking are important aspects of knowledge base debugging, but there is another aspect that has been largely ignored, that is determining whether the expert system will ask the user foolish or useless questions. This aspect of knowledge base debugging is important, because a system that asks the user foolish or useless questions will quickly lose credibility, and when the credibility of an expert system is in question, it will be abandoned by its users. To address this problem, we developed a procedure to help the knowledge engineer determine whether the system will ask the user foolish or useless questions. This procedure is an iterative process based on identifying conflicting question pairs. It has identified four constructions in rule-based expert systems that lead to inappropriate questioning. This procedure is effective, easy to implement, and has exhibited benefits, which are not confined to identifying constructions in the knowledge base that lead to inappropriate questioning. As a result, this procedure should become a useful tool that can be used by knowledge engineers to build better, more error free expert systems. We developed and tested the procedure on personal-computer-based expert systems; we are not sure how it will scale to main-frame systems.

1. Introduction

If a human tells a computer system that the subject is a male and the computer subsequently asks if the subject is pregnant, the computer system will lose credibility. Perhaps not even the worst of knowledge engineers would write a rule requiring those two questions. However, frequently the knowledge engineer is not as familiar with the knowledge domain as most readers are familiar with this example. Furthermore, these two questions might be triggered by rules such as the following that are widely separated in the knowledge base.

```
if sex = male and
. . .
then . . .
if pregnant = yes and
. . .
then . . .
```

0020-7373/91/120789 + 16$00.00/0

With such a construction, it is possible to ask the user whether a male subject is pregnant, unless the conclusion of the first rule precludes the second rule from being tested. It seems that such *lapses in logic* are possible in any knowledge base.

The consequences of such a "lapse" for the acceptance of the expert system may be devastating. It would be difficult to get a user to have confidence in a system that asked whether a male subject was pregnant. Even if the error were less obvious, this would still be an undesirable result because it would require the user to answer additional, unnecessary questions. This can be a source of frustration and may affect the user's attitude toward the system.

Expert system development has proliferated in the last few years due partly to the availability of inexpensive expert system shells for personal computers (Harmon, Maus & Morrissey, 1988). However, except for Bahill (1991), little attention has been paid to the issues of rule-base validation and verification. Following O'Keefe, Balci and Smith (1987) validation and verification are distinguished as follows. Validation refers to building the *right* system (that is, substantiating that a system performs at an acceptable level of accuracy), whereas verification refers to building the system *right* (that is, substantiating that a system correctly implements its specifications). This paper is concerned with verification.

Verification has traditionally been identified with consistency and completeness checking. Consistency checking consists of testing to show that a system produces expected answers (Cragun & Steudel, 1987). It includes checking for built-in discrepancies, ambiguities and redundancies in the rules of the knowledge base. It involves checking for redundant rules (e.g. two rules that succeed in the same situation and have the same conclusion), conflicting rules (rules that succeed in the same situation but have conflicting conclusions), subsumed rules (rules that have the same conclusion but one contains additional constraints on the situations in which it will succeed), rules with unnecessary conditions (rules that have the same conclusions and the same conditions except that one rule contains a condition that is negated in the other rule), rules that contain unreachable conditions (a rule in which there is no match for the condition either in the initial database or from a fact asserted by another rule) and circular rules (rules that, if taken as a set, form a cycle) (Nguyen, 1987; Jafar, 1989). Completeness checking tests to determine whether a knowledge base is prepared to answer all possible situations that could arise within its domain (Cragun & Steudel, 1987). Thus, completeness checking is used to find logical cases that have not been considered by the knowledge engineer resulting in missing rules. Although several papers have been presented to describe a general technique for addressing these issues (O'Keefe *et al.*, 1987; Cragun & Steudel, 1987; Nguyen, 1987; Jafar & Bahill, 1991), all these methods have been based on analysing the knowledge base directly and have not used knowledge from the expert to simplify the effort.

2. Development of our verification procedure

We wanted a verification procedure that would help the knowledge engineer avoid asking foolish or useless questions. To find such a procedure we first looked at traditional verification processes where the systems are run by the original expert and/or by other experts in the knowledge domain (Liebowitz, 1988). If, in running the systems, the experts encounter unnecessary or nonsensical questions, they bring them to the attention of the knowledge engineer. It is then up to the knowledge engineers to modify the knowledge bases to avoid these problems. We would like to design a procedure to automate this process. One major difficulty with the current method is that it depends on the expert (or others) exercising the knowledge base. It is possible that an instance of inappropriate questioning may slip by, because the expert seldom exercises all the rules in the knowledge base (Kang & Bahill, 1990). This is especially likely in a complex system, because the expert often focuses on specific cases, often those that are particularly difficult. As a result, the expert may miss problems that occur in cases that are not interesting. Second, this procedure might fail because the expert does not have time to test all possible cases or case types that the system is designed to handle. In any event, it must be a requirement of any process to automate this type of verification that it be capable of addressing or uncovering all instances of this problem.

Our procedure requires the expert's knowledge. The knowledge engineer may not have enough experience in the knowledge domain to identify conflicting or inappropriate questions. Our procedure cannot be based solely on the structure of the knowledge base, because the type of error that it is designed to detect does not qualify as a syntax error and consequently will not be exposed as a compile-time error. Finally, traditional verification algorithms cannot be used because this problem is not a problem with consistency or completeness of the knowledge base. What is needed, is a way of identifying questions that are inappropriate to ask under certain circumstances. Thus, we must identify both the questions and the conditions for asking questions in the knowledge base.

The approach taken in this paper was to develop a series of matrices that could be used to query the expert who was the original source of the knowledge. An associate would not be appropriate because the associate would not be aware of the subtleties in the knowledge base. The information acquired in this way could be used by the knowledge engineer to test the knowledge base.

The first step in the process was to create a matrix of all questions in the knowledge base, the question–question (QQ) matrix. Question identifiers were placed on each row and column of the matrix as shown in Figure 1. This gave a systematic way of making sure that all possibilities were checked. This approach is akin to the analytic hierarchy process, where pair-wise comparisons are performed and the results are put into matrix formats (Saaty, 1980). The advantage of this

QQ Matrix General Form

	q_1	q_2	q_3	...
q_1				
q_2				
q_3				
.				
.				
.				

FIGURE 1. The general form of the QQ matrix. The q_i (where $i = 1, 2, \dots, n$) are the question identifiers for question $1, 2, \dots, n$.

approach, as Saaty notes, is that it is easier to make pair-wise comparisons than it is to make a large number of simultaneous comparisons.

Once this matrix was constructed, the knowledge engineer could ask the expert to identify those question pairs that could not be redundant. Expanding our simple pregnant-male example, suppose that it was also necessary to determine the eye color of the subject. If q_2 was, "What is the sex of the subject?" and q_4 was, "What color eyes does the subject have?" then the expert might identify the (q_2, q_4) pair as questions that could not conflict regardless of how they were answered. Thus, both males and females could have blue, brown, green, or any other color eyes. In this case, a dash would be placed in the q_4 column of the q_2 row. An X would be placed in all boxes that could produce inappropriate questioning. Finally, the main diagonal is filled with zeros, because no question can be redundant with itself. Only the upper-right triangle of the matrix has to be filled out, because the matrix is symmetrical about the main diagonal.

Once the upper-right triangle of the QQ matrix has been filled out, the knowledge engineer would build a set of matrices for each of the "problematic" question pairs (the PQ matrices). Building this set of matrices involves two primary tasks.

First, the knowledge engineer must identify the hierarchy between each question pair to establish a "master" and "subject" question for each pair. This notion of a question pair hierarchy is intuitively simple but difficult to quantify. It is based on the way an expert organizes knowledge and the heuristic rules that are applied. To return to our simple example, we think that a question about the subject's sex should be asked before a question about whether the subject is pregnant. Therefore, the question about sex would be the master question and the question about pregnancy would be the subject question. Although establishing the master/subject hierarchy in question pairs is difficult to quantify, it is important to: (1) the flow of questioning; (2) the structure of the knowledge base; and (3) the use of the procedure outlined in this paper. The reason for this is that inappropriate and foolish questions in a consultation frequently result from incorrect coding of this relationship. As will be shown later in this paper, such coding errors include inappropriate rule order in the knowledge base and reversed premises in a rule.

Second, after the master/subject hierarchy is established for each question pair, the knowledge engineer, with the aid of the expert, must identify the *utilized values* for each "master" question of each pair. A utilized value for an expression can be defined as a value for an expression that causes at least one premise of at least one rule to be evaluated as "true". Utilized values are different than legal values. Legal values constitute the set of values for a given expression that are defined by the knowledge engineer to be acceptable responses to a question. Thus, if a user answers a question with a value other than a legal value, an error message should be displayed and the user should be asked to re-answer the question. By way of example consider the simple knowledge base shown below.

```
rule-1
if coat = hair
then type = mammal.

question(coat) = "What is the coat of the animal?".
legalvalues(coat) = (hair, feathers).
```

rule-2
if coat = scales
then type = fish.

"Hair" and "scales" are the utilized values for the attribute "coat"; whereas "hair" and "feathers" are the legal values for the attribute "coat". In a well-structured knowledge base, the legal-value set and the utilized-value set will be the same. But, it is possible, as this example proves, to include as legal values, values that are not utilized values, and to exclude utilized values from the legal-value set. In this example, the utilized value "scales" has been excluded from the legal-value set, and the legal value "feathers" is not contained in the utilized-value set. The requirement that the knowledge engineer identify all utilized values for each expression has the added benefit of forcing a comparison of the legal-value set with the utilized-value set for each expression. If the legal-value set is not equal to the utilized-value set, the knowledge engineer must resolve the inconsistencies before proceeding. Where an expression could have: (1) a numeric value (e.g. integer, real etc.); (2) a range of values (e.g. from 1 to 100); or (3) no legal values, appropriate utilized values or ranges of utilized values would have to be determined. For the rest of this paper we will assume that the conflicts between legal values and utilized values has been resolved, and we will use the term legal values for the result.

Because certain questions would be inappropriate only under certain conditions (e.g. it would be inappropriate to ask whether the subject was pregnant if the subject was identified as a male, but not if the subject was identified as a female), the PQ matrices have to include a listing of the legal values for the "master" question of each problematic question pair.

The PQ matrices, unlike the QQ matrices would not be square and would possess the general structure shown in Figure 2. The q_x (where $x = 1, 2, \dots, n$) are the question identifiers for "master" question and the lv_{xj} (where $x = 1, 2, \dots, n, j = 1, 2, \dots, m$) are the legal values for the "master" question, lv_{xj} being the jth legal value for the xth question, and the q_y (where $y = 1, 2, \dots, n$) are the question identifiers for the "subject" question.

Once these matrices are constructed the knowledge engineer asks the expert to identify answers (legal or utilized values) to the "master" question that would conflict with the "subject" question. If it would be inappropriate to ask question q_y when the response to question q_x was lv_{xj}, then an X would be put in the q_yth row and lv_{xj}th column. To use our simple example again, if q_2 was, "What sex is the subject?" and q_3 was, "Is the subject pregnant?" with legal values lv_{21} = "male" and lv_{22} = "female", an X would be made in the q_3th row at the lv_{21}th column. This would indicate that if the user answered "male" to q_2, then q_3 should not be asked. This is shown in Figure 3.

PQ Matrix General Form				
	q_x			
	lv_{x1}	lv_{x2}	lv_{x3}	. . .
q_y				

FIGURE 2. The general form of the PQ matrix.

Example PQ Matrix		
	What is sex?	
	male	female
Pregnant?	X	-

FIGURE 3. A specific example of a PQ matrix.

After this matrix is filled out, the knowledge engineer can use it to test the knowledge base. We investigated several techniques for applying this matrix; the best seemed to be modifying the knowledge base itself and then running the system. This is our heuristic for modifying the knowledge base:

2.1. HEURISTIC FOR MODIFYING THE KNOWLEDGE BASE

1. Instruct the shell that more than one value may be assigned to each expression (this includes expressions that do not appear in question statements). Typical statements for doing this are multi-valued, Limit and PLURAL.
2. Change the knowledge base so that no questions will be asked if a value for an expression is not found; for example, for VP-Expert you remove AUTOQUERY statements, for Guru you provide Find sequences without INPUT commands, while for M.1 you add "noautomaticquestion". This is necessary because testing is done non-interactively and we do not want the shell to generate a question and then wait for a human response. Next comment on the questions out of the knowledge base.
3. Set all expressions equal to all of their legal values. Where legal values have not been assigned to an expression, the expression should be set equal to all its utilized values. This requires a series of lines each of which assigns one of the utilized values for an expression to the expression. To use a simple example, suppose eye color has three utilized values, blue, brown, and green. Then this step would require three lines of code of the following form:

   eye color = blue
   eye color = brown
   eye color = green

4. Now the "problem" expressions have to be identified. Looking at the set of PQ matrices in our example, we would note a potential conflict between "sex = male" and the question about "pregnant". For these expressions undo step 1 above. That is make them single valued again (unless they were defined as multi-valued in the original knowledge base). Then, the value for the expression "sex" would be set to "male" and all other values would be excluded. All values for the expression "pregnant" would be excluded. That is, no value would be given to "pregnant" and only the value of "male" would be given to "sex".
5. With the knowledge base modified as described, a consultation would be run, and the intermediate results (cache) would be examined by the knowledge engineer.
6. When the consultation is complete, the process would return to step 4 and the next "problem" expression/question pair would be identified, making the necessary changes to the knowledge base to test that pair.

This process would be continued until all problem expression/question pairs were tested.

2.2. EVALUATING RESULTS OF OUR PROCEDURE

The output of our procedure might seem confusing to the knowledge engineer. The system may give nonsensical or contradictory advice. However, this procedure was not designed to test whether the expert system gives appropriate advice, but whether the system asks inappropriate questions. This can be determined by looking at the intermediate results (cache) from each of the consultations. All the knowledge engineer needs to do is scan the cache and see if a value for the inappropriate question's expression was sought. In our example, all that is needed is to determine whether a value for "pregnant" was sought when "sex" was set to "male". If it was, then it is known that the system would have asked the user whether the subject was pregnant after the user had indicated that the subject was male. By examining each saved cache, the knowledge engineer can determine if the system will ask inappropriate questions in any of the potentially problematic situations. If so, the knowledge engineer can act appropriately to rectify the problem.

3. Initial testing

Can our procedure be applied to knowledge bases in general and will it provide the knowledge engineer with any useful information about the structure and flow of the knowledge base? To test the usefulness of our procedure, we applied it to several simple knowledge bases by altering them according to the heuristic outlined above. The knowledge bases used initially were chosen because: (1) they were simple and would not be difficult to alter manually; (2) they could be analysed independently to determine whether our procedure flagged genuine errors; (3) they were representative of knowledge bases that are developed using personal computer-based expert system shells; and (4) the knowledge was simple and did not require sophisticated experts.

It should be noted that in the subsequent discussions of the implementation of our procedure on several knowledge bases, an attempt was made to identify the specific problem flagged by the procedure and correct it to see if our procedure would correctly identify that the problem had been corrected. It is not within the scope of our procedure to identify specific problems and identify a means by which the knowledge engineer can correct the error. Its function is only to indicate that a problem between question pairs potentially exists and it leaves it up to the knowledge engineer to identify why that question pair was flagged and how best to resolve the problem. Problem resolution was addressed only to make sure that our procedure identified genuine problems and to determine the type of problems our procedure could identify.

4. Development testing

Knowledge bases from three commonly used personal computer expert system shells were used. These include M.1 from Cimflex Teknowledge Corp., a rule-based system, VP-Expert from Paperback Software, which can be used either as a rule-based system or as an induction system, and 1stClass from Programs in Motion, an induction system. The initial trials included tests of two knowledge bases: the Animal Classification Expert System (Winston, 1977)† and Teknowledge's Cwine Expert System. VP-Expert and 1stClass were tested by using their induction systems to generate a knowledge base from an example set based on the "pregnant man" problem. Of these tests, only the one using the Animal Classification Expert System will be discussed in detail in this paper. This is done so that the reader can see how the procedure is implemented in the testing of a knowledge base. The test results of the other systems and the consequences of the results of those tests will simply be summarized in this paper. A discussion of the other tests can be found in (Bahill, 1991).

4.1. THE ANIMAL CLASSIFICATION EXPERT SYSTEM

In adapting this system, care was taken not to alter the structure of the knowledge base itself but simply to supply questions and other constructs that would make the knowledge base function in the M.1 environment. The following are the questions in the Animal Classification knowledge base:

q_1 = Of what material is the animal's coat composed?
q_2 = How does the animal move?
q_3 = Does the animal feed its offspring milk?
q_4 = What type of offspring does the animal have?
q_5 = What does the animal eat?
q_6 = What shape are the animal's teeth?
q_7 = What type of feet does the animal have?
q_8 = How are the animal's eyes positioned?
q_9 = What does the animal chew?
q_{10} = Where does the animal live?
q_{11} = What color is the animal?
q_{12} = How long is the animal's neck?
q_{13} = How long are the animal's legs?
q_{14} = What are the animal's markings?

These questions were used to generate the QQ matrix as shown in Figure 4. The zeros indicate that no question can be redundant with itself. The dashes indicate pairs of questions that the domain expert said could not conflict. And the X indicates the pair of questions that could cause inappropriate questioning.

The first four rules of the Animal Classification Expert System are as follows:

```
rule-1
if coat = hair
then type = mammal.
```

† The Animal Classification Expert System was chosen for this task because it is a simple, well-known knowledge base. Obviously it is not a good example of a problem that is amenable to expert system technology. However, it is an excellent heuristic example because people in general are familiar with the object, attributes and values, and knowledge engineers are even familiar with the rules and inferencing procedures. Therefore, they can focus on our validation techniques without learning new domain knowledge.

	QQ Matrix for the Animal Classification Expert System													
	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_{10}	q_{11}	q_{12}	q_{13}	q_{14}
q_1	0	-	X	-	-	-	-	-	-	-	-	-	-	-
q_2		0	-	-	-	-	-	-	-	-	-	-	-	-
q_3			0	-	-	-	-	-	-	-	-	-	-	-
q_4				0	-	-	-	-	-	-	-	-	-	-
q_5					0	-	-	-	-	-	-	-	-	-
q_6						0	-	-	-	-	-	-	-	-
q_7							0	-	-	-	-	-	-	-
q_8								0	-	-	-	-	-	-
q_9									0	-	-	-	-	-
q_{10}										0	-	-	-	-
q_{11}											0	-	-	-
q_{12}												0	-	-
q_{13}													0	-
q_{14}														0

FIGURE 4. QQ matrix for the Animal Classification Expert System.

rule-2
if feeds-offspring = milk
then type = mammal.

rule-3
if coat = feathers
then type = bird.

rule-4
if location = flies
and offspring = eggs
then type = bird.

These rules are used to determine the animal type (i.e., mammal or bird). Depending on whether the coat of the animal is "hair" or "feathers", the animal type can be determined without testing any other premise. The same can be said of the expression "feeds-offspring". That is, if the animal feeds its offspring milk, no other premise must be tested to conclude that the animal type is "mammal". Therefore, question-q_1 about the animal's coat and question-q_3 about what the animal feeds its offspring could generate a superfluous question, as is indicated in Figure 4. The clearest case in which the questions about the animal's coat and what the animal feeds its offspring would be redundant would be where the user, when asked about the animal's coat, answers that it is "feathers". In this case, the animal type would be concluded to be "bird" and asking what the animal feeds its offspring would not be necessary for determining the animal type.

The QQ matrix in Figure 4, shows that only questions q_1 and q_3 are potentially redundant. Most experts would probably make q_1 the "master" question and q_3 the "subject" question. Therefore, the PQ matrix shown in Figure 5 can be constructed.

From the PQ matrix for the Animal Classification Expert System, we can see that we need to test whether the user will be asked "if the animal feeds its offspring milk" after indicating that the animal's coat is feathers. The Animal Classification knowledge base was modified using our heuristic and the system was run. The

PQ Matrix		
	q_1	
	lv_{11}	lv_{12}
q_3	-	X

FIGURE 5. PQ matrix for the Animal Classification Expert System.
q_1 = Of what material is the animal's coat composed?
lv_{11} = the legal value hair
lv_{12} = the legal value feathers
q_3 = Does the animal feed its offspring milk?
The "X" identifies the legal value of q_1 that conflicts with q_3.

resulting cache was inspected; it was seen that a value for "feeds-offspring" was sought. This means that the user would have been asked "if the animal feeds its offspring milk" even after responding that the animal's coat was feathers. So we know now that this knowledge base was constructed so that it asks at least one inappropriate question of the user. The next question for the knowledge engineer is, "How can the knowledge base be modified so that it will not ask inappropriate questions of the user?"

By default, most rule-based expert system inference engines, test rules sequentially from the top of the knowledge base to the bottom. Thus, the first rule in the knowledge base is tested first, the second is tested second and so on until the last rule is tested. The rule involving "feeds-offspring" occurs before the rule that tests to see if the animal's coat is feathers. Therefore, in an attempt to fix the appropriate question problem, we interchanged rule-2 and rule-3 and ran our procedure again. This time no value was sought for "feeds-offspring" and thus, the system would not have asked the user this inappropriate question.

What does this exercise tell us about our procedure? First, it can be used to identify when a knowledge base might be constructed so that an inappropriate question could arise. Second, it has specifically identified a situation where the rule order of the knowledge base leads to this type of problem. Third, it has demonstrated that changing the order of the rules in the knowledge base can solve the problem.

4.2. THE CWINE EXPERT SYSTEM

Our procedure showed that in the Cwine Expert System it would be a mistake to ask the user about the type of sauce if the meal did not have sauce.

The knowledge base was altered using our heuristic, the system was run and the results of cache were evaluated. It was found that the question about the type of sauce would be asked even if the user indicated that the meal did not have sauce. In analysing the knowledge base it became clear that the rules were constructed, so that it would be impossible to rearrange them, as was done with the Animal Classification Expert System, to prevent the system from testing the rules associated with the expression "sauce". Teknowledge anticipated this problem and developed the "presupposition" statement to rectify it. The presupposition statement allows the knowledge engineer to stipulate the conditions under which a value for an expression should be sought. This could also be handled with a "screening clause." A screening clause is a premise that determines whether or not a value will be sought for an expression in a subsequent premise. In the examples below, if the value of has-sauce is "no", then the user will not be asked if the sauce is cream.

Knowledge base with a screening clause;
if has-sauce = yes
and sauce = cream
then best-color = white.

Knowledge base with a presupposition:
presupposition(sauce) = has-sauce.
if sauce = cream
then best-color = white.

In testing the Cwine Expert System, our procedure correctly identified the potential problem of the system asking the user what kind of sauce the meal has after the user said that the meal does not have sauce. However, our procedure did not indicate when the problem had been fixed using a presupposition statement. Because, if M.1 determines that the expression has-sauce is no, it concludes that sauce is unknown, and does not bother to ask the user for a value for "sauce". Thus, "sauce" is always sought although the shell does not necessarily ask the user to assign a value to it.

4.3. TESTING ON INDUCTION SHELLS

It is reasonable to assume that our procedure could be extended to other personal computer rule-based systems. However, there is another class of expert system shell commonly used on personal computers, namely, induction systems. The question naturally arises, "Can the problem of developing a knowledge base that by its construction asks the user inappropriate or foolish questions be developed in the induction environment, and if so, will our procedure prove useful in that environment?"

To answer this question two commonly used personal computer expert system shells were tested, VP-Expert and 1stClass. We constructed a simple set of examples to test whether a "logically-flawed" knowledge base could be induced by these systems. We used our "pregnant man" paradigm to construct our set of examples. Although most people are aware of the problem of asking whether a male is pregnant, this is just a simple example of a more general problem. For example, we suspect that it would be more difficult for most people to decide whether it was inappropriate to ask whether a *Trimusculus reticulatus* (mollusk species) was a *Tadarida brasiliensis mexicana* (bat subspecies). (It would be.)

Trials with both VP-Expert and 1stClass demonstrated that it was possible to generate a "logically-flawed" knowledge base through injudicious choice of examples. This result is not surprising, because a rule (for VP-Expert) or a branch of a decision tree (for 1stClass) is generated for each example. Consequently, if these shells are given an inappropriate example they will generate an inappropriate rule or branch. Although this result may seem trivial, the trials with VP-Expert and 1stClass proved helpful in identifying additional constructs that effect inappropriate user questioning. These include the importance of the position of premises within a rule and possibility, especially in an induction system, of generating incorrect rules. Details of these tests can be found in (Bahill, 1991).

4.4. RESULTS OF THE DEVELOPMENT TESTING

Our procedure can identify four constructions that can lead to inappropriate questioning of the user. These constructions can be rectified by the knowledge engineer in one of the following ways:

1. rearranging the position of a rule in the knowledge base, as shown for the Animal Classification Expert System,
2. rearranging the premises within a rule, e.g. write

```
IF sex = female
AND pregnant = yes
THEN result = expanded_waistline.
```

not

```
IF pregnant = yes
AND sex = female
THEN result = expanded_waistline.
```

3. inserting screening clauses in the problem rules or using a presupposition statement, as described in our Cwine example, and
4. removing incorrect rules.

5. Implementation testing

In the development of our procedure outlined in this paper we used simple knowledge bases to test our procedure and determine the type of errors that it might identify. Next it was necessary to test our procedure on more sophisticated knowledge bases. Initially, it was felt that the best source of knowledge bases for testing our procedure would be found in the 80 expert systems that were generated by Systems and Industrial Engineering students at the University of Arizona as projects for a course on expert systems. Unfortunately, most of this source had to be abandoned because the procedure depends heavily on acquiring information from the knowledge domain expert, and an expert was available for only one of these expert systems, VWMod, that was designed to provide advice on modifying a Volkswagen 1600 cc engine to optimize performance.

A second source of knowledge bases include three expert systems that were initially built as class projects, were later expanded as Masters projects, and are currently being prepared for commercial release. In each of these cases, the original domain expert was available and was willing to participate in our testing.

Stutter is an expert system designed to help with the diagnoses and prognosis of children who may have begun to stutter. Of its 595 possible question pairs, 45 were identified as "problematic" question pairs. Because the master question for some of these pairs contained multi-valued expressions, it was determined that 187 tests would have to be run to test all the possible "unaccepted-value sets" generated by those questions. This at first appeared to be an insurmountable task, especially because our heuristic was being implemented by hand, but it was noticed that most of the master questions included several subject questions. That is, it was determined that if the master question was answered with one of its "unaccepted-value sets" several of its subject questions should not be asked. This allowed testing several question pairs with only one alteration of the knowledge base. Therefore, the number of tests needed dropped from 187 to 71. So, although it was difficult and time consuming, it was possible to test all 45 problematic question pairs with only 71 alterations of the knowledge base.

Chromie is an expert system that was designed to help teach medical professionals, who are not experts in chromosomal abnormalities, how to identify the abnormality of an unborn or newly born baby to allow them to make an intelligent referral. Similar to Stutter, Chromie was originally constructed as a student project in 1985, was later the subject of a Masters project and is currently undergoing testing in preparation for its commercial release. Chromie, with a total of 60 questions, provided a staggering 1740 possible question pairs. Fortunately, of those only 67 were identified as potentially problematic. Unlike Stutter, none of the master questions in the problematic pairs contained multi-valued expressions. However, as with Stutter, several master questions contained multiple subject questions and this allowed several question pairs to be tested with one alteration of the knowledge base.

When all the tests of Chromie had been run, our procedure flagged 62 of the 67 problematic question pairs as containing potential errors. Of these potential errors, 24 had been anticipated by the knowledge engineer and were handled in the knowledge base using presupposition statements. The remaining 38 had not been anticipated and came as somewhat of a surprise to the knowledge engineer.

In trying to verify that our procedure had correctly identified potential problems in the Chromie knowledge base, an inherent weakness in our procedure was identified. Although our procedure identified several problems that could lead to inappropriate questioning of the user, it cannot identify what the problem is. Our procedure can determine that some combination of user inputs can lead to inappropriate questions but it cannot identify which combinations can. The first question that we were asked when we informed the knowledge engineer that our procedure has detected several potential errors between problematic questions pairs was, "What responses led to the problem?" Unfortunately, without examining the knowledge base this question cannot be answered by our procedure. Thus, the procedure shows that inappropriate questioning is possible, but does not indicate how it is possible.

It also should be mentioned that it is possible that our procedure has identified "unexpected" user questioning rather than "inappropriate" user questioning in the Chromie knowledge base. In testing the system the knowledge engineer and the expert have been running test cases that determine how they will answer questions. As it turns out, many problems flagged by our procedure resulted from the use of "unknown" (identified as a legal value). Thus, some of the resulting questioning may be appropriate under those circumstances. It remains for the knowledge engineer and her expert to decide whether they want the system to ask additional questions of the user under these circumstances and how this "unexpected" user questioning affects the conclusions reached by the system.

The importance of these results is that the procedure identified several potential problems in the Chromie knowledge base, whether they are instances of inappropriate or unexpected user questioning. It is up to the knowledge engineer to determine the impact of these potential problems.

FundEye is an expert system designed to help with the diagnoses of retinal disease. Similar to Stutter and Chromie, it was originally developed as a student project, and is now being prepared for commercial release. Initially, FundEye appeared to pose a significant problem in implementing our procedure. Similar to Chromie, FundEye contained a large number of questions (80), which yielded a total of 3160 question pairs that would have to be evaluated. Because of the large number of question pairs that would have to be evaluated, it was feared that the task would require a great deal of the expert's time and, thus, might prove to be impractical. To our surprise, the initial screening of the questions (developing the QQ matrix), required only a little over one hour to complete. Part of the reason for this swiftness is that many questions could never conflict. For example, questions about patient history like the patient's name, and the doctor's name could never conflict with other questions in the knowledge base. This initial screening identified only 21 "problematic" question pairs. When all the tests of FundEye had been run, 13 possible inappropriate questionings were flagged. Twelve of these could be fixed with screening clauses, and the other could be fixed by rearranging the premises.

These expert systems were all relatively large for personal computer-based expert systems. We doubt that brute force enumeration could be effectively used on these systems. Our technique has used the knowledge of the expert to reduce the search space. Frankly, we were surprised that this technique worked, but it did.

The implementation testing has shown that in three of the four knowledge bases tested, our procedure would identify potential conflicts between question pairs (see Table 1). Furthermore, most of these conflicts could be rectified in one of the ways outlined earlier. For example, several of the conflicts detected in the VWMod, Chromie, and FundEye knowledge bases could be rectified by adding "screening clauses" to the problem rules. At least one conflict in the FundEye knowledge base resulted from inappropriate premise ordering and could be rectified by simply re-ordering the premises in the problem rule.

6. Other potential mistakes detected in verifying these systems

The benefits of implementing our procedure are not confined to the identification of the constructions described above, however. Our procedure has been helpful to the knowledge engineer and expert in identifying problems that were not expressly tested for by our procedure. These include problems with question phrasing, redundant questions, inconsistencies between "utilized" and "legal" values, inconsistencies between the premises of rules and expressions that have numeric values or that do not have assigned legal values and questionable or suspect rule structures that do not have a direct effect on user questioning.

Detection of these problems results from our procedure's requiring the knowledge engineer and the expert to re-evaluate the knowledge base from a different perspective. Knowledge bases are generally constructed in stages with the knowledge engineer and the expert concentrating their efforts on a particular class of problems during each stage. This can lead to a fragmented understanding of the overall structure of the knowledge base. Our procedure forces the knowledge engineer and the expert to look at the entire system through the mode of user questioning. This requirement and the need to modify the knowledge base in

TABLE 1

Results of implementation testing of our procedure

Knowledge base	File size (Kbytes)	Number of questions	Number of question pairs	Tested pairs	Time spent altering and testing knowledge bases (hours)	Time to make the QQ and PQ matrices (hours)	Number of potential inappropriate questionings detected	Number of other potential mistakes detected
VWMod	27	16	120	3	3·5	0·5	2	3
Stutter	83	35	595	45	12·5	1·0	0	14
Chromie	96	60	1,740	67	12·3	0·8	62	0
FundEye	72	80	3,160	21	6·8	2·0	13	27

preparation for testing, can lead both the expert and the knowledge engineer to uncover problems in the knowledge base that might otherwise go undetected.

Modifying the knowledge base in preparation for testing, may uncover problems that result from having: (1) some utilized values that are not assigned as legal values (this error can lead to the failure of some rules to fire correctly or to having rules fire unexpectedly); (2) having multiple questions for a single expression (this error may cause changes in question structuring not to be implemented as expected); and (3) omissions in the premises of rules (this error may lead to inappropriate user questioning as well as causing a rule to fire inappropriately).

We have found that having the knowledge engineer and the expert discuss the relationship between various questions in the knowledge base, often enabled the expert to identify redundant questions (which require the user to answer more questions than are necessary in a consultation), vague or poorly phrased questions (which do not really address the distinctions that the expert wants to make), and generalized questions that need to be broken into multiple specific questions. Implementing our procedure outlined in this paper, also allows the expert to rethink some of the rules of thumb that the knowledge engineer has coded into the knowledge base. As a result of this "rethinking" the expert may uncover conflicts in thinking or may be able to establish new relationships that may increase the accuracy of the advice ultimately given by the system.

This procedure can accommodate incremental enhancements of the knowledge base. If a new question is added to the knowledge base, a new row and column are added to the QQ matrix. Entries in this new row and column must be checked against the other questions in the knowledge base, but the rest of the matrix remains unchanged. Further, because the PQ matrices depend only on identified problematic question pairs, the existing set of PQ matrices does not need to be altered although additional matrices may need to be added depending on the evaluation of the new question in the QQ matrix.

7. Limitations

In this paper we only considered *pairs* of questions that could lead to inappropriate questioning of the human. It is possible to encounter situations where the conjunction of two questions would preclude intelligent query of a third. For example, let

q_1 = What is the city?
q_2 = What is the date?
q_3 = What time is sunset?

If the answer to the first question is Prudhoe Bay, and the answer to the second question is June 21, then the third question should not be asked.† This quagmire could be extended to cases where, if three questions have certain answers, then a fourth question should not be asked. We have not tried to deal with this situation. We have been told that the theory of colored graphs may provide tools for dealing with such problems, but we suspect that even for personal computer-based expert systems this may be intractable.

† The sun never sets on June 21 in Prudhoe Bay because it is above the Arctic circle.

The order in which questions are asked is sometimes important. Our concept of master–subject relationships could help establish the most logical order. But, in general, our procedure does not deal with question order.

8. Conclusions

This paper has presented a verification procedure that can be applied to personal computer rule-based expert systems. It allows the knowledge engineer to correct bugs, and refine the knowledge base. It has been shown that this procedure can be implemented on relatively sophisticated expert systems without an inordinate drain on the expert's time or the resources of the knowledge engineer. It has identified four common errors in rule construction that led to inappropriate user questioning. However, the benefits of implementing our procedure are not confined to the identification of these constructions. Our procedure has been shown to be helpful to the knowledge engineer and expert in the identification of problems that are not expressly tested for by our procedure. These include problems with question phrasing, redundant questions, inconsistencies between utilized and legal values, inconsistencies between the premises of rules and expressions that have numeric values or that do not have assigned legal values, and questionable or suspect rule structures that do not have a direct effect on user questioning. Because of the many benefits of implementing our procedure and because its implementation has not proven to be insurmountable, our procedure should be a useful tool to help the knowledge engineer build better, more error-free expert systems.

References

Bahill, A. T. (1991). *Verification and Validation of Personal Computer Based Expert Systems*. Englewood Cliffs, NJ: Prentice-Hall.

Cragun, B. J. & Steudel, H. J. (1987). A decision-table-based processor for checking completeness and consistency in rule based expert systems. *International Journal of Man–Machine Studies*, **26**, 633–648.

Harmon, P., Maus, R. & Morrissey, W. (1988). *Expert Systems Tools & Applications*. New York: Wiley.

Jafar, M. J. (1989). A tool for interactive verification and validation of rule based expert systems. University of Arizona, *Ph.D. dissertation*, Department of Systems and Industrial Engineering.

Jafar, M. J. & Bahill, A. T. (1990). Validator, a tool for verifying and validating personal computer based expert systems. In D. E. Brown & C. C. White, Eds. *Operations Research and Artificial Intelligence: The Integration of Problem Solving Strategies*. pp. 373–385, Boston: Kluwer.

Kang, Y. & Bahill, A. T. (1990). A tool for detecting expert system errors. *AI Expert*, **5**(2), 46–51.

Liebowitz, J. (1988). *Introduction to Expert Systems*. Santa Cruz, CA: Mitchell.

Nguyen, T. A. (1987). Verifying consistency of production systems. In *Proceedings of the IEEE 3rd Conference on Artificial Intelligence Applications, Kissimmee, FL*, pp. 4–8.

O'Keefe, R. M., Balci, O. & Smith, E. P. (1987). Validating expert system performance. *IEEE Expert*, **2**(4), 81–90.

Saaty, T. L. (1980). *The Analytic Hiearchy Process*. New York: McGraw-Hill.

Winston, P. H. (1977). *Artificial Intelligence*. Reading MA: Addison-Wesley.