An Expert System for Off-Line Analysis of Rotating Equipment

by

James R. Hoglund

Thesis submitted to the faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree

of Master of Science

in

Mechanical Engineering

APPROVED:

Dr. R. Gordon Kirk

Dr. Larry D. Mitchell

Dr. Douglas J. Nelson

August, 1989
Blacksburg, Virginia

An Expert System for Off-Line Analysis of Rotating Equipment

by

James R. Hoglund

(ABSTRACT)

The analysis of rotating equipment difficulties is currently accomplished by a specialist in rotating equipment examining the signs and symptoms of the equipment, applying his expert judgement, and determining the cause of the machine's difficulty. This thesis covers the development of an off-line expert system that can be used to emulate the expert's ability to interpret the signs and symptoms of the machine, including suggestions of possible further actions to take for repairing the problem or refining the proof for the proposed cause the difficulty. An editor has also been built which will allow an expert to keep the information used by the system current with the state of the art for rotating machinery diagnostics. This thesis documents the development of the PC-based Turbo Prolog expert shell and external knowledge-base editor. The use of the system editor is illustrated by loading current diagnostic table information into a knowledge base. Then the expert program's operation is

illustrated by applying the editor-formed database in a typical session. The expert shell and knowledge base can operate as a stand-alone unit for field application. Resident experts in machinery diagnostics can build and update databases for distribution to users in the company to assure full uniform utilization of the current and most correct knowledge.

Acknowledgements

I would like to thank the chairman of my committee, Dr. R. Gordon Kirk, for his help and guidance on this endeavor. I would also like to thank Dr. Larry D. Mitchell and Dr. Douglas J. Nelson for serving on my graduate committee.

I would also like to thank my father, , and my two brothers, , for their continual support of my research effort. I would also like to pay special tribute to my late mother, who insisted I remain in graduate school during her terminal illness.

Special thanks also go out to who has explained how to use the Total Word editing package for this thesis writing. Using his free time on my behalf has made final completion much easier.

Table of Contents

1.0 INTRODUCTION.....1

1.1 The Problem.....2

1.2 The Solution.....3

1.3 Earlier Efforts.....3

2.0 PROLOG.....5

2.1 Predicates and Pattern Matching.....5

2.2 Backtracking.....6

2.3 Forward and Backward Chaining.....8

2.4 Flexible Internal and External Databases.....9

3.0 ELEMENTS OF EXPERT SYSTEMS.....10

3.1 Domain.....10

3.2 Reasoning and Search Techniques.....11

3.3 Data Characteristics.....12

3.4 User Interface.....12

3.5 System Maintenance.....13

4.0 EXPERT SYSTEM.....14

4.1 External Database Structures.....16

4.2 Editor Program.....19

4.2.1 Preparing Information for Database Construction...21

4.2.2 Using the Editor to Form the Database.....24
4.3 Expert Shell.....33
4.3.1 Computational Logic.....35
4.3.2 Using Expert Shell.....37
5.0 CONCLUSIONS AND RECOMMENDATIONS.....43
REFERENCES.....46
Appendix A. Rockland Chart.....48
Appendix B. Predicates and Code for Editor.....51
Appendix C. Predicates and Code for Expert Shell.....90
VITA.....112

List of Figures

1. Basic Components of An Expert System.....15
2. Editor Program Flow Chart.....20
3. Choose Database Screen for Editor.....25
4. Rule Edit Screen for Editor.....27
5. Rule Parts Screen for Editor.....28
6. First Condition Screen for Editor.....29
7. Second Condition Screen for Editor.....30
8. Expert Shell Flow Chart.....34
9. User Prompt Screen for Expert Shell.....38
10. Solutions Screen for Expert Shell.....40
11. Conditions Used to Prove a Rule in Expert Shell....41
12. Condition Not Known Screen in Expert Shell.....42

Nomenclature

OCF	rule's original confidence factor
MOCF	rule's present confidence factor
$\sum$CDK	current sum of the used "Don't Know" weighting factors
n	number of conditions in a rule
n_N	current number of "No" responses
$\sum$CN	current sum of the used "No" weighting factors

Chapter 1

INTRODUCTION

Major industries typically have an expert or a group of experts to solve the many vibration problems that occur in rotating equipment. Many times, the problems are more numerous than the expert staff can manage. For this occurrence, it would be beneficial to have an expert system that would solve for the more common cases and aid people present at remote sites in their diagnostic efforts.

After examining available shells and programs already developed for expert system creation (1,2), it was decided to develop a custom shell in Turbo Prolog. After determining the abilities of the language, it was found to be suitable for development of a commercially useful expert shell.

1.1 The Problem

Rotating machinery diagnostics has been attempted by a method of observing signs and symptoms, and then applying these results against known patterns related to various machine difficulties. Well-known tables by Sohre (3), Jackson (4), and Rockland (5) are good examples of the sign and symptom diagnostics. Unfortunately, these tables are inherently static, unless a user personally chooses to modify them. But if the tables were greatly lengthened and improved with additional diagnoses, signs, and symptoms, along with possible fixes, related standards, methods of evaluating machine behavior, and other useful information, they would soon become very unwieldy. As a result, matching diagnosis patterns through long tables of possible diagnosis would become so tedious that people using the table would possibly miss a better diagnosis. What is needed is a concise method of organizing and manipulating large amounts of this type of information in a way that encourages use by making it easy to update and utilize.

1.2 The Solution

With the continual decrease in the cost of fast and powerful computers, and with the advent of more complex computer languages, the ability to create programs to solve difficult problems has continually grown easier. The obvious advantage is that computers will perform as instructed without error, and can manipulate vast quantities of information in a short time. In recent years a new group of programs, called expert systems, have evolved to solve problems where algorithm-only solution processes are too inefficient or unable to find solutions do to incomplete knowledge. These new programs use heuristics to guide solution methods and seek out possible solutions to non-deterministic answers.

1.3 Earlier Efforts

Many attempts at developing expert systems for rotating machinery may be found in the literature. However, the first commercial code was not available until 1985, when the Radian Company produced Turbomac (6). Radian observed that most of the diagnostic codes and monitoring codes used were not usable by the majority of people who are concerned with machinery operation. Of the

systems being developed in the literature, most are attempting to create systems that both monitor and analyze continually the machine's performance. This is a costly set-up for the plant, and is limited to the ability of the models to simulate the current behavior of the machine.

Before this thesis was done, no commercially produced expert system was designed specifically for the companies that want off-line expert systems to aid with diagnosing rotating machinery difficulties.

Chapter 2

PROLOG

Specialty computer languages, called Artificial Intelligence languages, have been developed to try to handle logical problems by the use of heuristic operations needed for solving problems without one specific answer. The most common of these are LISP and PROLOG. Being these two languages both have great strengths for this purpose, Borland's Turbo Prolog 2.0 was chosen on the basis of cost, overall capability, and ease of distribution to potential users. This language provides many benefits that aid in creating the expert system. These are explained in the sections below.

2.1 Predicates and Pattern Matching

Prolog programs are built from a starting point, the goal, to an end statement, similar to most other

languages. The main difference is that almost every command in the main program line, called predicates, is written by the programmer. This is similar to most all lines being subroutines. The other difference is that Prolog attempts to satisfy predicates, not solve them. This is best explained by a simple example. Assume the predicate takes two positive integers, five and two, and adds them up to yield a third number, seven. The only way this predicate would work in most computer languages is to send in a five and a two and get back a seven. In Prolog, the predicate could be sent any two of the numbers and would return the third. So if the predicate were sent the sum, seven, and one of the numbers, two, it would return five, the number that satisfies the relationship. If the relationship queried about had an infinite number of solutions, like what two numbers added together equal seven, the predicate would return the first solution it found for the relationship. If no solution could be found, then the predicate would fail.

2.2 Backtracking

When an error occurs in most programs, they stop execution. In Prolog, the program starts going back through the lines of code, in the opposite order in which

they were executed, to the last place where it could have chosen a different response. In the above example, assume the predicate following the add predicate is the product of the two numbers equals ten. On first pass, Prolog would return for the two numbers whose sum is seven, the numbers one and six. It would then use these responses to try to solve the second predicate. Being that their product doesn't equal ten, the second predicate would fail. The program would then backtrack to the most recent predicate where it could have made a different choice. In this case it would go to the predicate for two positive integers equal to seven. This time it would choose a different possible solution, two and five. Now it goes forward to the next predicate and test to see if their product is ten. Since this is true, the program would continue on to the next predicate in the program. This behavior of backtracking to try to find alternate solutions is what allows Prolog to behave in what appears to be an intelligent way. In Prolog, the programmer describes what the system restrictions, and the desired end points for the solution. Then the program finds a way to satisfy the goal given the starting and ending points specified by the program's user.

2.3 Forward and Backward Chaining

Forward and backward chaining are methods of finding solutions to problems. In forward chaining, all known information is loaded into memory at the onset of the analysis. Then the program determines what conclusions can be made from the known information. Backward chaining takes a possible conclusion and tries to prove that the conclusion is true by proving its parts. These two solution methods can both be used in solving problems with incomplete information. Well written chaining methods would store missing links to final answers, or query the user to determine whether the missing links had known values. A final result would then include possible answers and what assumptions had to be made in order to arrive at those answers. For off-line systems, it is more logical for the program to use backward chaining. This is because data entry is not automatic, but is done by user interaction with the code. If the user had to answer fifty or so questions before the program could attempt to seek solutions, then the program would appear unwieldy to the user, and would not be used.

2.4 Flexible Internal and External Databases

One of the advantages of Prolog is its ability to manipulate databases both resident in memory and on disk. Although most languages have the ability to read and write to disk files during operation, Prolog is able to change their contents very easily, and can treat them in much the same way as it does predicates. Assume the programmer has a data file that contains the makes, cost, and age of a set of pumps that are for sale. The user could specify a capacity, price range, oldest desirable age, and undesirable makes, and then the program could return from the database all entrees that satisfied the user's specification. Although this could be done in many languages, very few could do it all with five commands, including the output statement. Most would have to loop through the database once for every entree in the database. But Prolog, through the use of backtracking, could do it all in one loop.

Chapter 3

ELEMENTS OF THE EXPERT SYSTEM

Wolfgang, Dear, and Galbraith (2) describe five groups containing a total of nineteen traits that all expert systems should contain. Many of these traits are provided by using Prolog. Others are a product of programming methods. And some are descriptive of the way the program is used and maintained. These shall be reviewed by their respective groups, and discussed as relevant to the expert system developed.

3.1 Domain

The information contained in the expert system should be for a specific area of expertise. This is satisfied by creating and using databases that pertain only to rotating machinery.

3.2 Reasoning and Search Techniques

The expert system uses heuristic reasoning to manipulate strings that represent facts about the problem under consideration. This power is given to the system by Prolog, which is one reason that an AI language is used in creating expert systems.

All responses that the user gives that are important, and all conclusions created by the expert system are stored in databases operating in active memory. This way the decision as to what reasons to pursue is a product of previous responses.

The expert system uses the power of Prolog to seek solutions by pattern matching known user responses to the possible answers in the external database. By doing so, the expert system tries to predict possible reasons as to why the machine has difficulties.

The expert system uses recursion to exhaustively search its database for possible solutions. This way it generates a list of all possible reasons for the difficulty. To order these possible solutions from most to least likely, the program uses some form of computational logic that utilizes certainty factors. These factors are weighed according to user responses to give the final certainties associated with every possible reason of machine difficulty.

3.3 Data Characteristics

The data representing the possible reasons for machine difficulties is stored in a easy to modify form. Most of the information is stored in strings and in reference numbers to strings created by rotating equipment experts. These strings describe the possible machine difficulties and their conditions, and other useful information for particular difficulties. Testing on one reason stops when it becomes too unlikely, or is found to be a potential reason of the difficulty. Testing then starts on the next reason. The shell has special handling techniques built in to deal with incomplete information, and stores the facts that are incomplete. This way it can be determined later what unknown conditions were used to prove which difficulty reasons.

3.4 User Interface

The expert system asks questions of both a qualitative and quantitative nature. Any time that a user accesses the conclusions screen, a full selection of summaries concerning results of the program's search are available. Most user interactions are of the form of full questions with a list of possible word responses.

3.5 System Maintenance

Basic databases are to be created from standard references. It is hoped that by making these databases easy to edit that they will be maintained and updated by the controlling expert with regularity. The format of the shell gives individual companies the ability to load their company proprietary and unique information, references, and standards into databases. The editor has been provided to allow users without knowledge of Prolog to supplement the current knowledge base provided or create new ones for specific products. The editor and expert solver are not connected so that the expert can provide current database copies to novices without worrying about them entering incorrect data into the database.

Chapter 4

EXPERT SYSTEM SHELL

The basic components of an expert system are shown in Fig. 1. Please note that in this system, the inference engine is a part of the expert shell. The primary focus of the work has been the development of the inference engine, editor, and the ability to output the suggestions, fixes, and references for possible solutions. The contents of the stored databases are only briefly dealt with so that the strength of the editing system may be demonstrated. The discussion shall be done in three sections; first, a discussion of the external database structures; second, being the structure and use of the editing system, and third, the structure and use of the expert shell. When combined with databases created from the knowledge of experts in rotating machinery diagnostics, a fully operational expert system is formed.

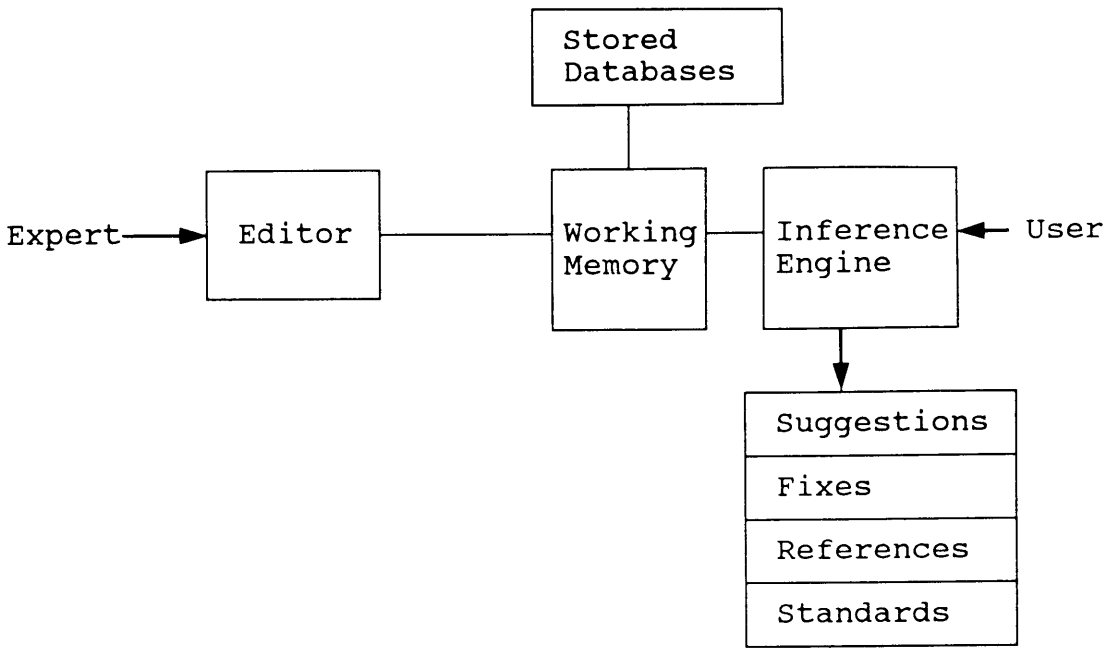

Figure 1 Elements of an Expert System

4.1 External Database Structures

The program uses five substructures in its external databases. Of greatest importance is the rule structure. The other rule structures are conditions, references, suggested further actions, and standards.

The rule data structure has seven parts. Rules are the reasons for the rotating machine's malfunction. There will only be one rule for each difficulty reason. The structure is

```
rule  (REFERENCE_NUMBER,  MAXIMUM_CONFIDENCE_FACTOR,
       CONDITION_LIST, RULE_DESCRIPTION, REFERENCE_LIST,
       SUGGESTED_REPAIRS_LIST, RELATED_STANDARDS_LIST).
```

The reference number is an integer that gives the rule number. This is a unique number that differentiates a rule, or cause of malfunction, from every other rule. The maximum confidence factor is a real number describing the maximum confidence that the rule can obtain. Values between one and zero represent the range from one hundred to zero percent, respectively. The conditions list is a list of condition numbers followed by their corresponding loss of confidence factors. These factors are the weighting factors for conditions being untrue and not known. The common value in both cases is one, representing full appropriate loss of value for such a response to a condition. A zero represents no loss in the overall confidence for a no or maybe response, as

specified. The rule description is a string that is the descriptor of the reason for the difficulty that the rest of the rule is describing. The reference, suggested repairs and related standards parts are lists of the reference numbers of the related references, related actions and repairs that can be done for a diagnosed difficulty, and the related standards for the rule, respectively.

The second data structure is the conditions data structure. Conditions are the signs and symptoms exhibited by the various possible machine difficulty reasons. The structure of this database is

```
cond   (REFERENCE_NUMBER,   CONDITION_DESCRIPTION,
     HOW_TO_CHECK).
```

This structure is made of a unique reference number for the condition, followed by two strings. The first string is the condition description. This is a description of the sign or symptom. The second string is a description on how to test for the presence of the sign or symptom.

The third data structure is for references. References are included so that a non-expert can easily find relevant material to a difficulty, like previous similar cases. The basic structure of a reference is

```
library (REFERENCE_NUMBER, AUTHOR, TITLE, PUBLISHED_BY,
        PUBLISHING_DATE).
```

The reference number is a unique integer representing the reference. The other entries are strings containing the

author's name and the reference's title, publisher, and publication date.

The forth data structure is for suggested actions in the event of a solidly diagnosed difficulty. This has been included so that phone numbers or routine responses for simpler problems can be suggested when a successful diagnosis is reached. The structure is

more_to_do (REFERENCE_NUMBER, SUGGESTED_ACTION).

The reference number is the unique identifying integer for each database entry. The suggested action is the string that represents the action that needs to be done by the user in the advent that they feel they have the correct reason for the machine's difficulty.

The fifth data structure is for related standards. This is included so that standards for maximum allowable vibration levels for a particular machine or difficulty are easily found when a diagnosis has been made. The general structure is

```
standard  (REFERENCE_NUMBER,  FIRST_HALF_OF_TITLE,
          SECOND_HALF_OF_TITLE,  STANDARD'S_SOURCE,
          PUBLICATION_DATE, SUMMARY_LINE, SUMMARY_LINE
          SUMMARY_LINE)
```

The reference number is the unique integer assigned to each standard. The rest of the elements are strings for the standard's title, its source, the date of publication, and a short three line summary of the standard.

4.2 Editor Program

This editor creates and modifies the external databases that are used by the expert shell. A special editor is necessary because the form of the database is not readable as normal ASCII characters. The editor does not allow the removal of conditions, references, standards, or suggestions of possible actions from the database. This is to avoid an accidental removal of a condition that is still in use by other rules.

Figure 2 is a flow chart of the editor. Note that unlike flow charts for other languages, ones for programs in Turbo Prolog must include path lines for predicate successes and failures. Also, the program has several methods to reach some points on the flow chart, and some options will not always be accessible. The flow chart may be pictured as showing the right path lines for as long as the user is always choosing appropriate actions. If the user is not always choosing good options, his requested action is not necessarily going to be obeyed. Also, branching doesn't always happen at set spots, and backtracking can go quite a distance in the program in some cases. Title_page, run, part_2, and part_3 are the actual predicate calls used in the program. Title_page is a page of text where introductory information about what the program does is provided for identifying the purpose

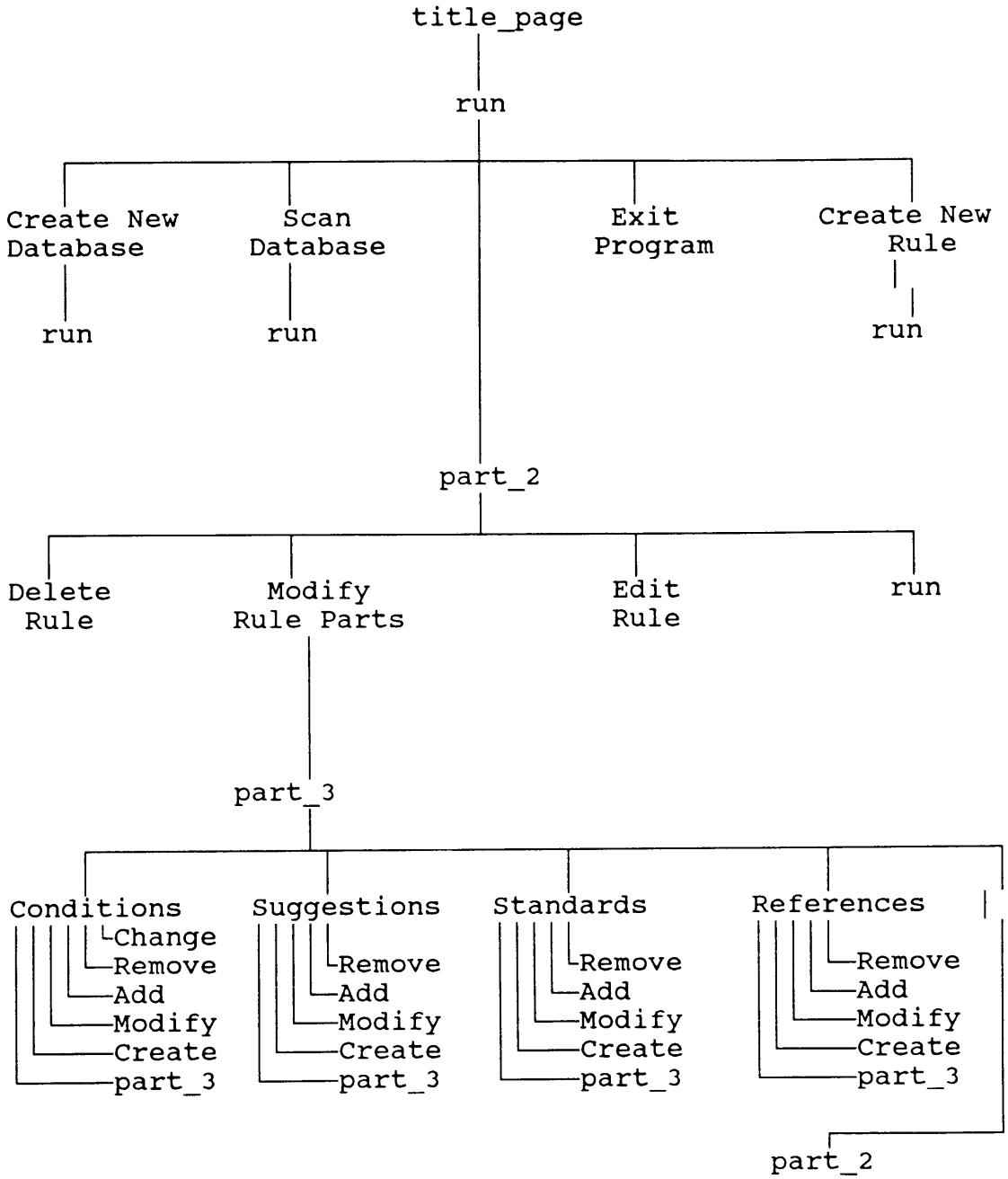

Figure 2: Flow Chart for Editor

of the program. Run, part_2, and part_3 are predicates that present menus that give options for the branching below them in the flow chart, and then execute appropriate actions to start those actions happening.

4.2.1 Preparing Information for Database Construction

The best demonstration of the editor is to use it to convert one of the published trouble-shooting tables into a database. The Rockland Chart is a very simple and straight forward table and so the first three entries in the Chart (Appendix A) will be used in this example. The first line is for a probable cause of mass unbalance. Examining the third box, note that the dominant plane of vibration is different for centerhung and overhung machines. This difference is continued in the phase angle relationship. Therefore, these two cases will be considered as two different rules. Both rules will share the traits of a disturbing frequency of 1x rotor speed, steady amplitude of vibration, and a narrow band vibration peak on log paper. The traits that are different are dominant planes of vibration and the phase angle relationships. These traits are the conditions for the two rules. There are no related standards, suggested repairs, or references given in this table, so these

sections remain blank. So, the rules are

Rule : Mass Imbalance (centerhung machine)
Conditions :
1. this machine is center hung
2. the disturbing frequency is 1x rotor speed
3. the radial direction vibration is dominant
4. the inboard and outboard vibration are in phase
5. the vibration amplitude is steady
6. has a narrowband vibration envelope

Rule : Mass Imbalance (overhung machine)
Conditions :
7. this machine is overhung
2. the disturbing frequency is 1x rotor speed
8. the axial direction vibration is dominant
9. the inboard and outboard vibration is 90 deg. out of phase
5. the vibration amplitude is steady
6. has a narrowband vibration envelope

At this point, the table's knowledge on mass imbalance has been exhausted. Notice that several conditions, 2, 5 and 6 have been repeated. This is desirable as it makes the database more compact. Also, two machine characteristics, centerhung and overhung, appear as conditions. This is because these two physical differences change some of the signs observed. Now, repeating this process for the second and third line of the table results in two rules for both lines. This is a result of the disturbing frequency entry for bent shaft and the amplitude comment for line three. Repeating the above process results in the following four rules.

Rule : Bent Shaft (at midspan)
Conditions : 2. the disturbing frequency is 1x rotor speed
8. the axial direction vibration is dominant
10. the inboard and outboard vibration is 180 deg. out of phase
5. the vibration amplitude is steady
6. has a narrowband vibration envelope

Rule : Bent Shaft (at coupling)
Conditions : 11. the disturbing frequency is 2x rotor speed
8. the axial direction vibration is dominant
10. the inboard and outboard vibration is 180 deg. out of phase
5. the vibration amplitude is steady
6. has a narrowband vibration envelope

Rule : Eccentric Motor Mass (no electrical problem)
Conditions : 12. the machine is a motor unit
2. the disturbing frequency is 1x rotor speed
13. the disturbing frequency is 1x and 2x line frequency
3. the radial direction vibration is dominant
5. the vibration amplitude is steady
6. has a narrowband vibration envelope

Rule : Eccentric Motor Mass (electrical problem also)
Conditions : 12. the machine is a motor unit
2. the disturbing frequency is 1x rotor speed
13. the disturbing frequency is 1x and 2x line frequency
3. the radial direction vibration is dominant
14. the vibration fluctuates or beats in amplitude
6. has a narrowband vibration envelope

4.2.2 Using the Editor to Form the Database

In the preceding case, fourteen conditions have been used to prove five rules. The next step is to load these results into a database. The first step in activating the editor is to load it into an IBM Personal Computer by typing in the name of its executable file. The first screen to appear identifies the program and the version number. The next screen introduces what the program is and gives some background information. After pressing return again, the next screen (Fig. 3) is bright green on the top, and presents a menu of options on the lower half of the screen. This top half will remain bright green until a database has been loaded into memory by creating a new one or loading an old one from disk. Since the purpose is to create the Rockland database, F1 is pressed and Rockland is then typed in as the name of the new database.

Now that the database has been created, it needs to contain rules. Rules are added to the database by pressing F3 for the add a new rule to the database option. Pressing F3 yields a request for the descriptor of the rule. The name of the rule should be typed in, followed by F10 when the name is correct. Next the program ask for the maximum confidence factor for the rule. In this case, the Rockland Chart does not address the likelyhood for any

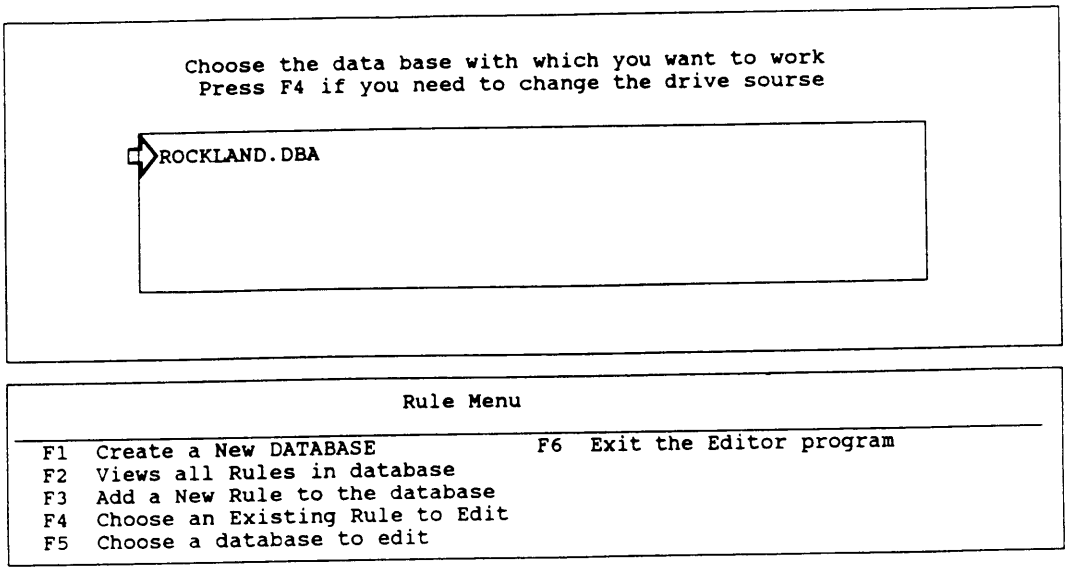

Figure 3: Choose Database Screen for Editor

of the rules, so a default value of 0.90 will be entered for all rules.

After all the rules have been entered into the database, it is time to assign conditions to the rules. This is done by placing the light bar over the first rule, and pressing F4 to edit this particular rule. Pressing F4 yields a new menu (Fig. 4) at the bottom of the screen containing editing options for the rule. The intent is to add conditions to the rule, so F2 is pressed.

At this point another new menu appears on the screen (Fig. 5). This menu gives the options of adjusting any of the rule parts that have separate database forms designated for them. F2 through F4 are parts not covered by the Rockland Chart, and so will not be used in this example. The first option, to edit the conditions, will be activated by pressing F1.

Now the second of two condition menus appear (Fig. 6). These two can be differentiated by the background color of the upper window. This screen, which shows the conditions currently in the database, is back lighted in black. The first menu (Fig. 7), has the list of all conditions assigned to the rule back lighted in blue. This second condition menu is normally accessed by pressing F3 on the first menu for the option to add or create conditions for the rule. Being there are no conditions assigned to this rule yet, and no conditions present in the database,

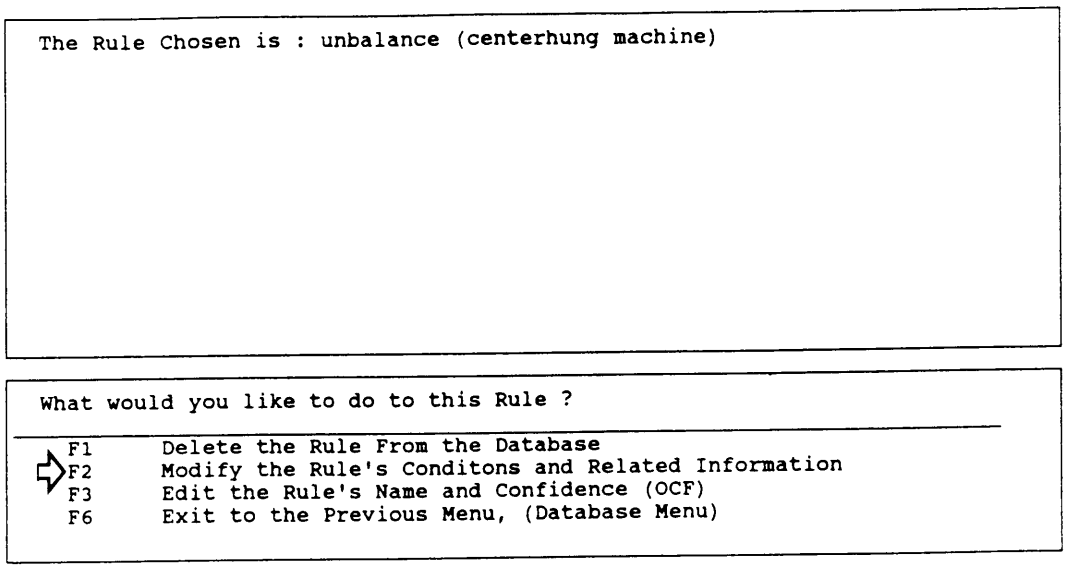

Figure 4: Rule Edit Screen for Editor

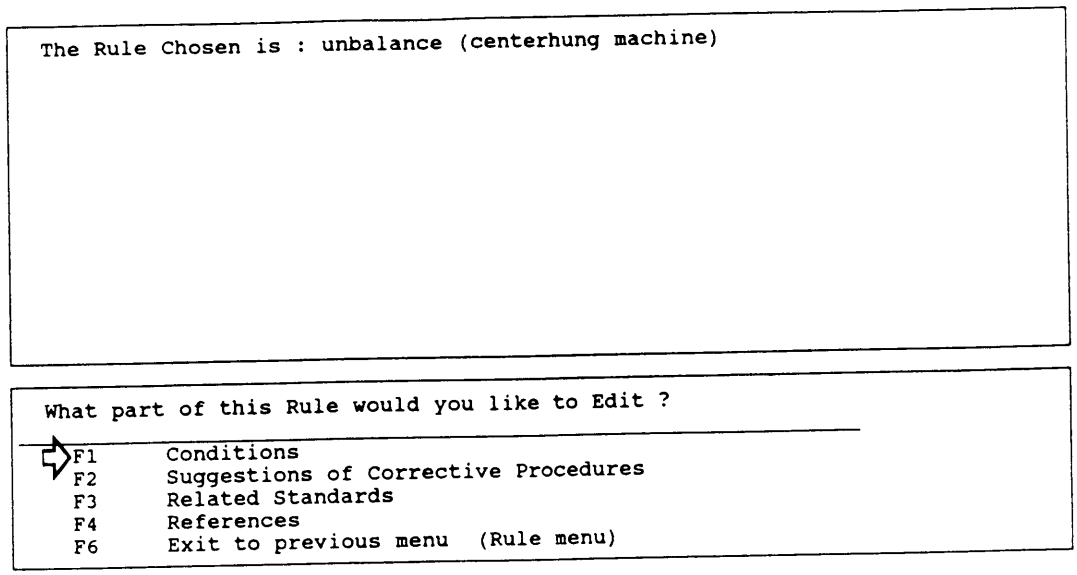

Figure 5: Rule Parts Screen for Editor

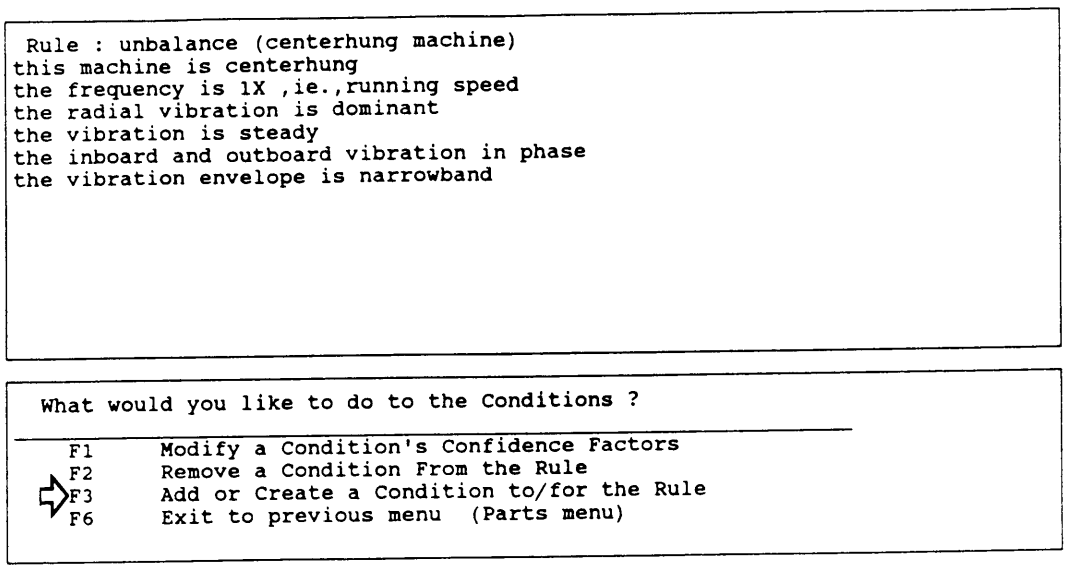

Figure 6: First Condition Screen for Editor

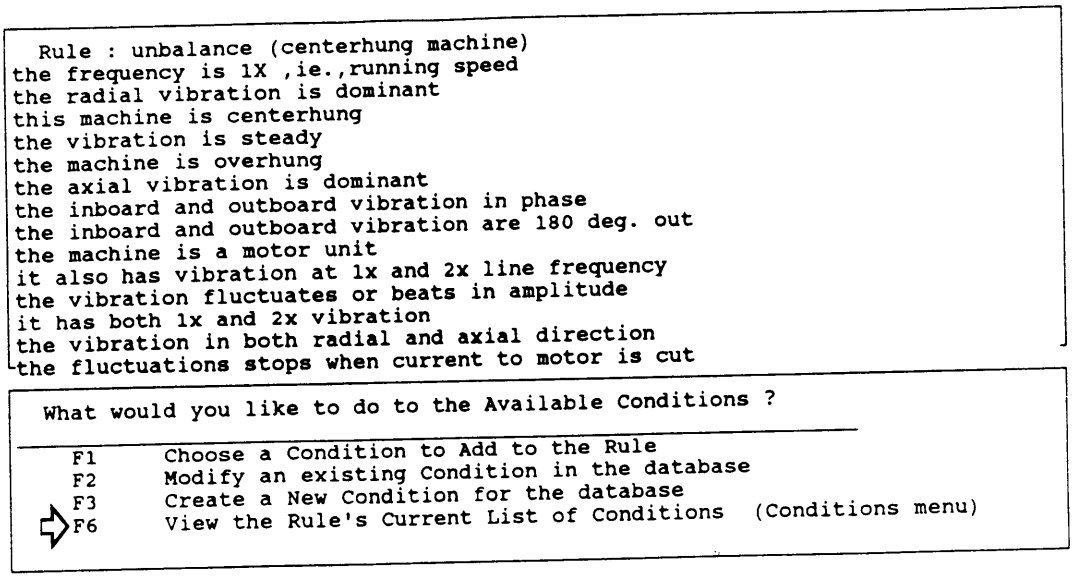

Figure 7: Second Condition Screen for Editor

the program goes straight to the create conditions option.

The create a new condition option, F3, on the second menu as normally encountered, first prompts the user for the condition's explanation. This is the name of the condition, or the entry found in the Rockland Chart. After the user presses F10, they are prompted for the condition's description. This is a short statement on how the condition is identified or where it is found. This is a clue to the non-expert as to how to find the condition, in case he doesn't know how to test for it or recognize that particular phrasing for the condition. Being the list of all needed conditions for the first six rules is known, all conditions can be created for the database by continual use of F3.

After all the conditions have been created, it is time to assign the current rule its conditions. This is done by placing the light bar over the condition needing to be added to the rule, and pressing F1 on the second condition screen. The program will ask for the loss of confidence factors for No and Don't Know responses. These values will all be one, except for no responses for physical parts like oil seals that have to be present for some difficulty reasons. These will be assigned a loss of confidence factor of two for "No" answers so that the absence of these parts will automatically eliminate those rules from consideration. After each condition is added,

the program returns to the first menu so that the user can view the current list of conditions assigned to the rule. So, each addition of a condition will take the pressing of F3 on the first menu for the add/create option, and F1 on the second menu for adding a condition present in the database to the current rule.

After all six conditions have been added to the first rule, conditions need to be assigned to the rest of the rules. If any references, standards, or suggested repairs were suggested by the table, then additional editing on the rule would be done by going to the screen offering rule parts for editing and choosing the other part forms that need to be added. But being the Rockland Chart doesn't indicate any information of this type, F6 needs to be pressed repeatedly until the menu offering the choice to edit a rule or choose a new database to edit. Here, the next rule needs to be picked up and taken to the condition menus to have conditions attached. This is done until all rules have had all their conditions assigned to them.

If this is repeated for all the lines of the Rockland Chart, a database is built that is ready to be used by the expert shell for difficulty analysis, but has none of the potential advisory functions that the shell allows.

4.3 Expert Shell

The function of the expert shell is to use the information stored in a database along with responses from the user to determine a list containing the probable reasons of the machine's difficulty in descending order of likelihood. Figure 8 shows a flow chart of the editor. Title_page, run, restart, search, prompt, and output are actual predicates used in the expert shell. Title_page is a few pages of text that introduce the expert shell. Run and restart determine the database whose knowledge is going to be used in the judgement and how unlikely a solution will be considered by the program. Search controls the programs efforts to prove rules. Prompt controls the question and answer dialog between the computer and the user. It also uses the computational logic scheme described in the computational logic section to determine if a rule is too unlikely to still consider as a possible cause of difficulty. Output does the presentations for the conclusions of the expert shell. It includes solutions and how it concluded each solution, along with useful information that is related to each solution.

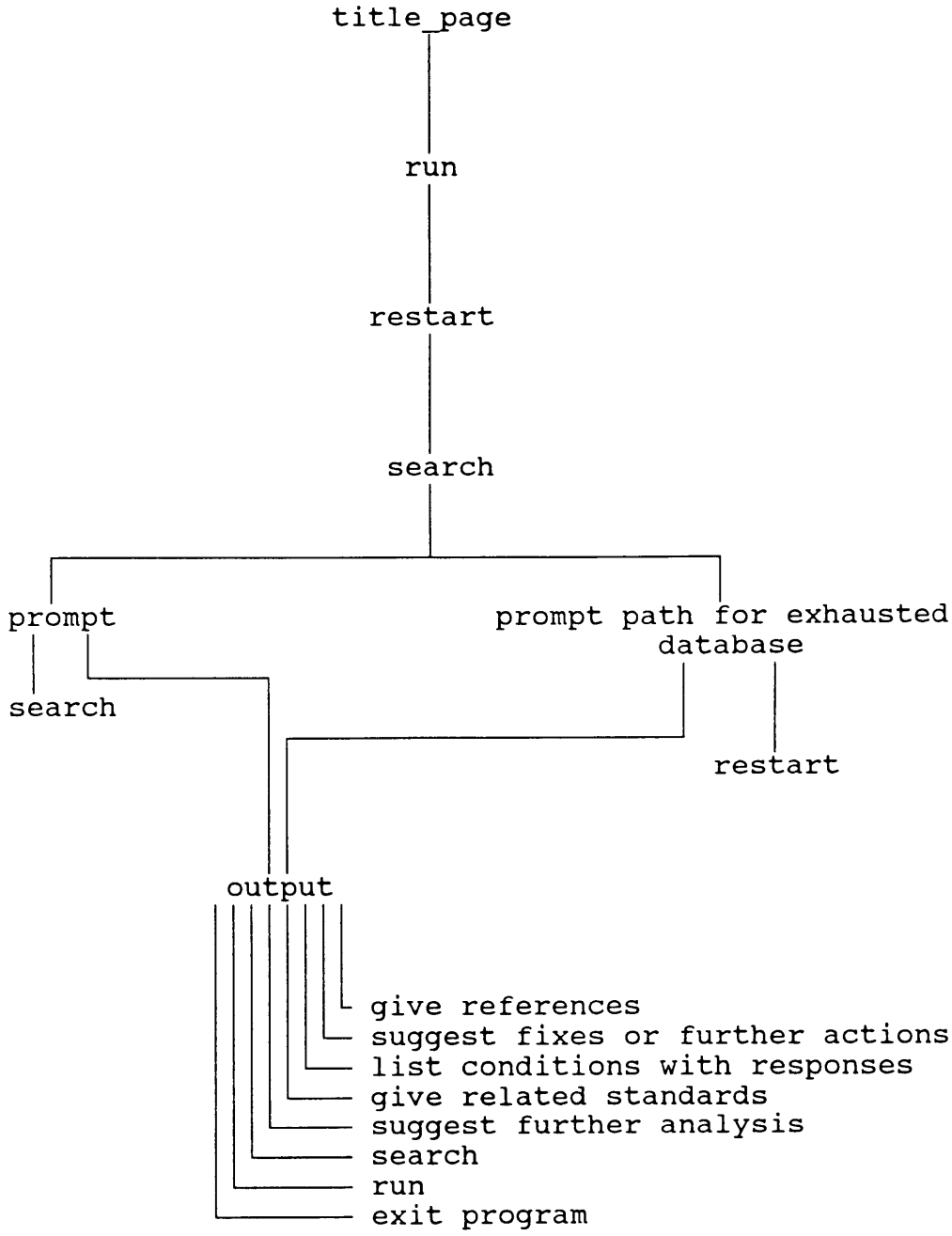

Figure 8: Flow chart of Expert Shell

4.3.1 Computational Logic

The expert shell requires some way to determine if a rule is still worth pursuing as a possible cause of rotor difficulty. It also requires some method to determine the rankings for reasons of difficulties. This is all accomplished by some scheme of numbers that assign probability by weighting schemes using user responses and a general value for the rule. The names for these various schemes include fuzzy logic (2), Dempster Shafer approach (7), and Bayes rule and equation (8). All of these assume a certain probability function applies to a particular difficulty in all cases in which it might be applied. Being no two machines by different companies are the same, and many models by the same company are often evolving between models or are unique builds, such assumptions are not well founded. In light of this, it was decided to create a simpler scheme that allowed for several possible user responses, general confidence factors for rules, and the ability to weight various conditions more or less heavily.

The resulting scheme is that each rule has a unique machine expert set value called an original confidence factor (OCF) that represents the highest probability that a rule is the cause of the machine's current difficulty. For each condition, the user is allowed responses of

"Yes", "No" and "Don't Know". Yes responses cause no loss in the rule's confidence. Each Don't Know response causes a second order increasing loss in rule's confidence with each response. For each successive No response the rule suffers a half order rate loss of confidence. Each of rule's conditions have two factors associated with them in case the loss in confidence for a No or Don't Know response is deemed to strong or mild. If the current confidence of the rule drops below a user defined value, it is eliminated as not being a likely solution. The equation to determine the modified original confidence factor (MOCF) is

$$MOCF = OCF - [OCF \times \frac{\Sigma CDK}{n+1}]^2 - [OCF \times \frac{n_{N\_}}{n-1}]^{1/2} [\frac{\Sigma CN}{n_N}] \quad (1)$$

All rules that have a final confidence greater than the cut-off value are stored in memory and displayed on the solutions screen in descending rank of MOCF. For a rule with an OCF of 0.9 and five conditions of an equal weighting of one gives the following MOCF values for the following combinations of Yes (Y), No (N), and Don't Know.

Y	N	Don't Know	MOCF
5	0	0	0.90
4	0	1	0.88
3	0	2	0.81
2	0	3	0.70
1	0	4	0.54
0	0	5	-0.05
4	1	0	0.43
3	2	0	0.23
2	3	0	0.08
1	4	0	-0.05

4.3.2 Use of Expert Shell

The Rockland database has been built with all the information of the table, plus explanations for conditions added by Dr. R. G. Kirk. This resulting database has no standards, references, or suggested repairs attached to it, so while the options in the expert shell work and are available, they will not be demonstrated. Excerpts from a possible session will be given to demonstrate more closely how the program operates. On figures, arrows have been added when choices are being made on menus because the colored light bar does not show up on the diagrams.

The program is started by placing the source disk into an IBM-PC and typing in the proper name for the executable code. The first window encountered identifies the program and gives the version number. After two screens of introductory information, the computer displays the list of available data files on the disk. After identifying the data file from which judgements will be rendered, the program then ask for the minimum allowable confidence factor that will be considered. For more refined information, higher values can be used to speed up the search. The value chosen for this case will be 0.3.

Now the program starts the question and answer session part its operation. These screens are of the form of Fig. 9. The menu gives the three responses that are available

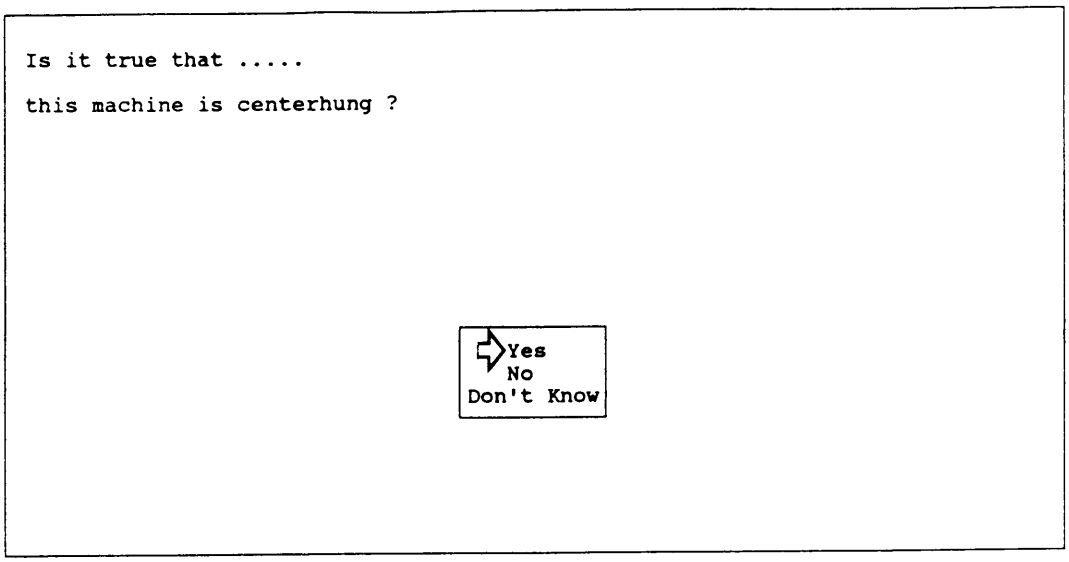

Figure 9: User Prompt Screen for Expert Shell

to the user. After an answer has been found by the expert shell, the user has the option of continuing the search or going to view the solutions that have been found.

The solutions are presented in Fig. 10. The found solutions are listed from greatest to least confidence in the upper half of the screen. The confidence factors have been included so that users who want to make use of them can have them readily available. In case the number of possible solutions found by the program is greater than six, the list of solutions is capable of being scrolled. The lower half of the screen list the many options of information that are available to the user. All information is available for printing if the user chooses, as are a few commands to allow the user to start a new or continue the current diagnostics session. It is often useful to have hard copies of the way a rule was solved (Fig. 11). This way the group of conditions and the way they were answered will explain the program's reasoning. It is also useful to firm up a diagnosis by checking for conditions to which the user didn't know if they were present. A typical case is shown in Fig. 12.

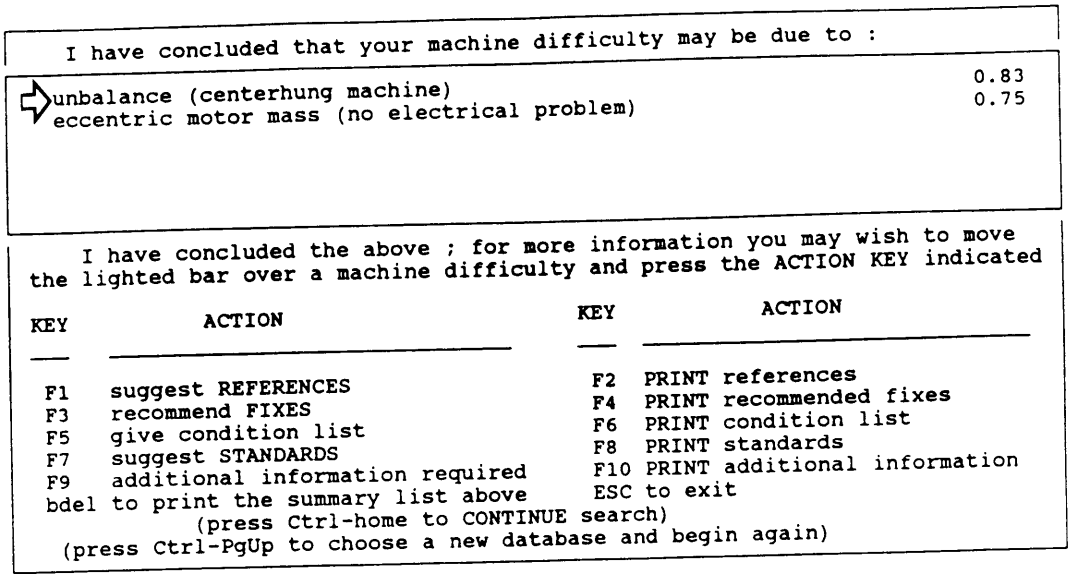

Figure 10: Solutions Screen for Expert Code

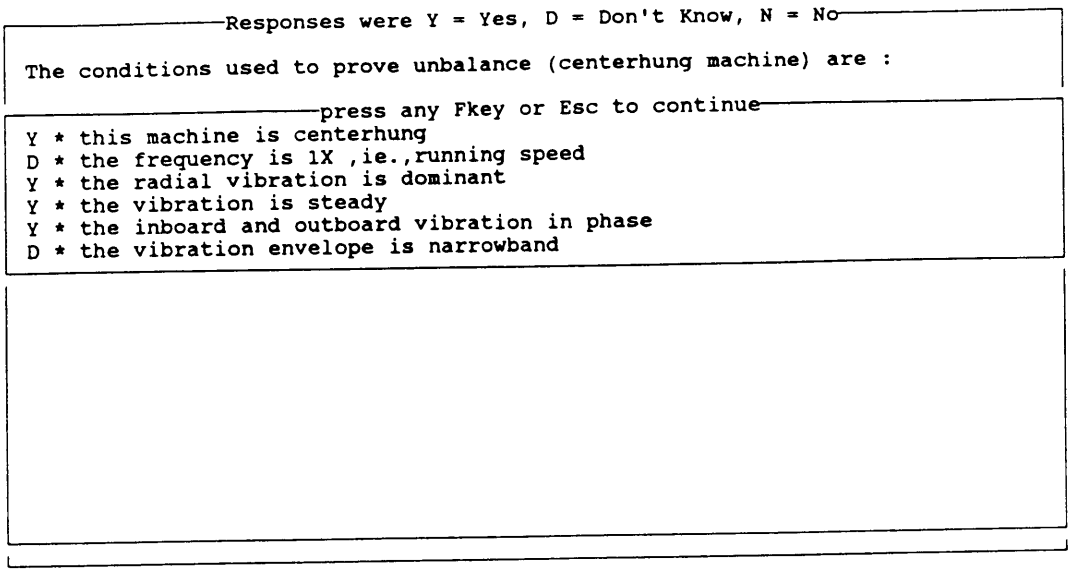

Figure 11: Conditions Used to Prove a Rule in Expert Shell

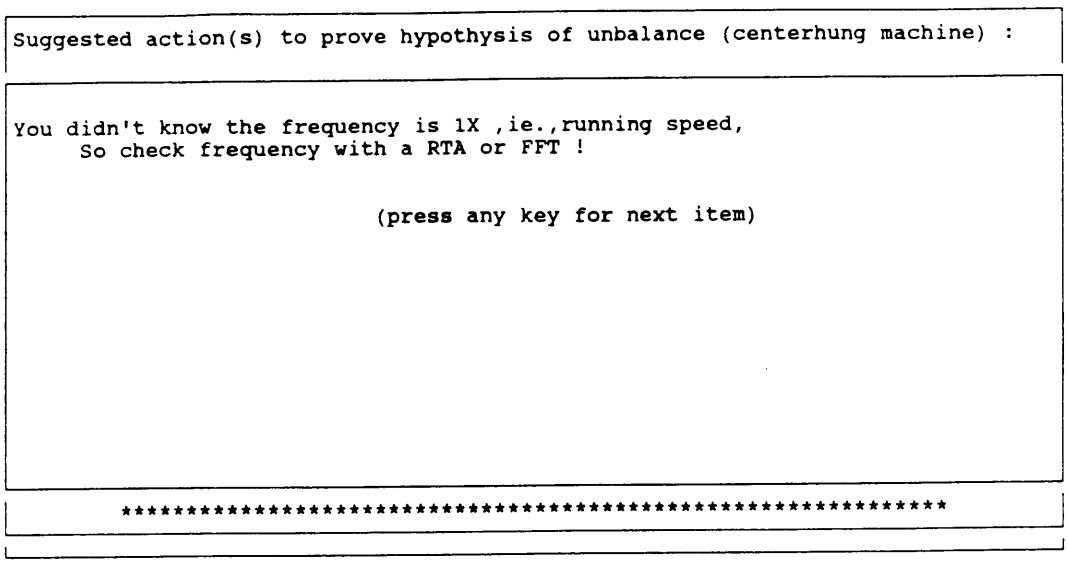

Figure 12: Condition not Known Screen in Expert Shell

Chapter 5

CONCLUSIONS AND RECOMMENDATIONS

From the results presented in the thesis, the following conclusions have been reached:

1. A commercially usable expert shell can be built and operated on an IBM-PC for use in rotating machinery diagnostics.

2. Both internal and external databases may be utilized in the shell. Currently, only external databases have the capacity to contain large amounts of information as is found in well developed databases.

3. Resident experts can build and update databases for distribution to users in the company to assure full uniform utilization of the current and most correct knowledge.

4. Although the maximum size obtainable for a database has not been determined, by the claims of Borland who make Turbo Prolog, it should be adequate for application to special equipment types. So far, the

largest database tested, of this format, has contained about 30 rules and 80 conditions.

Considering the results of this research and these conclusions, the following recommendations can now be made:

1. Specialized databases should be developed for various classes of rotating machinery, such as motors, steam and gas turbines, cooling fans, expanders, process pumps, and boiler feed pumps. These should be based primarily on the open literature, so that references can be made to the material and its use there for not restricted.

2. A utilities package for the editor should be assembled containing delete functions for conditions, suggested further actions, references, and standards. It should also include options to compress databases, copy information from database to database, and prepare various printed forms of the contents of databases.

3. The expert shell should be tested in an industrial environment to refine the methods for weighting factor selection and extended database construction.

4. On-line capability is necessary to permit the program to operate as a machine monitoring and diagnostics expert package. This would also allow the program to test for qualities that are hard to describe to people who are not rotating machinery experts. This will require the

program to be modified to allow a forward chaining analysis and communication with hardware like vibration probes and real time frequency analyzers.

REFERENCES

1. Watermann, Donald A., A Guide to Expert Systems, Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.

2. Wolfgang, Deborah D., Teresa J. Dear, and Craig S. Galbraith, Expert Systems for the Technical Professional, John Wiley and Sons, New York, 1987.

3. Sohre, J. S., "Operating Problems with High-Speed Turbomachinery--Causes and Corrections," ASME Petroleum Mechanical Engineers Conference, Dallas, Texas, September 1968.

4. Jackson, C. J., The Practical Vibration Primer, Gulf Publishing Company, Houston, Texas, 1979.

5. "Machinery Vibration Diagnostic Guide," Rockland Scientific Corporation, Rockleigh, New Jersey, 1988.

6. Stuart, J. D., and J. W. Vinson, "Turbomac: An Expert System to Aid in the Diagnostics of Cause of Vibration-Producing Problems in Large Turbomachinery," Radian Technical Report No. ST-RS-00968, presented at ASME Conference, Boston, Massachusetts, August 4-8, 1985.

7. Zimmersmann, Hans J., Fuzzy Set Theory - and its Applications, Kluwer-Nijhoff Publishing, Boston, Massachusetts, 1985.

8. Shafer, Glen, Mathematical Theory of Evidence, Princeton University Press, Princeton, New Jersey, 1976.

9. Pearson, Carl E., Handbook of Applied Mathematics, Selected Results and Methods, Van Nostrand Reinhold Company, New York, 1983.

10. Turbo Prolog 2.0 Reference Guide, Borland International, Scotts Valley, California, 1988.

APPENDIX A

ROCKLAND CHART [5]

Reproduced from reference 5 with permission from Rockland Corporation. TEL (201) 767-7900 or FAX (201) 767-1890

MACHINERY VIBRATION DIAGNOSTIC GUIDE

PROBABLE SOURCE	DISTURBING FREQUENCY	DOMINANT PLANE	PHASE ANGLE RELATIONSHIP	AMPLITUDE	ENVELOPE CHARACTERISTICS	COMMENTS
UNBALANCE (1) Mass imbalance	1x rotor speed	radial (Axial is higher on overhung rotors)	1 Force in phase (0°) 2 Couple out-of phase (90°)	steady	narrowband	Rotor looseness or bow due to thermal stresses may change the amplitude and phase with time
(2) Bent shaft	1x rpm (2x rpm if bent at the coupling)	axial	180° out-of phase axially	steady	narrowband	Run-out at rotor mass appears as unbalance. Run-out at coupling appears as misalignment
(3) Eccentric motor rotor	1x rpm, 1x and 2x line frequency	radial	N/A	steady (see comment)	narrowband	Will fluctuate (beat) in amplitude and phase if an electrical problem also exists (see electrically induced)
MISALIGNMENT (1) Parallel	1x, 2x rpm	radial	radial 180° out-of-phase	steady	narrowband	Most misalignments will be a combination. Errors are most common in the vertical plane. On long coupling spans, 1x will be higher
(2) Angular	1x, 2x rpm	axial	axials 180° out-of phase			
(3) Both parallel and angular	1x, 2x rpm	radial and axial	Both radial and axials will be 180° out-of-phase			
ELECTRICALLY INDUCED All electrically caused problems can be isolated, i.e. eliminated by cutting the current to motor (1) Eccentric rotor	2x slip frequency sidebands around 1x rpm, 1x and 2x line frequency	radial	N/A	steady		Will fluctuate in amplitude and phase if mechanical problems (imbalance) also exist
(2) Loose stator laminations	2x line frequency and high frequency (60K CPM) sidebands of 2x line frequency	radial	N/A	high, steady	narrow band sidebands of 2x line frequency	Not usually destructive
(3) Broken rotor bar	running speed with 2x slip frequency sidebands	radial	N/A	steady	narrow spike with sidebands	Replace rotor bar
(4) Unbalanced coil or phase resistance	2x line frequency	radial	N/A	low, steady	narrowband	
(5) Stator problems (heating, shorts)	2x line frequency	radial (can cause axial)	N/A	steady	narrowband	Vibration will increase as motor heats up
(6) Loose iron	2x line frequency	radial	N/A	high, steady	narrowband	
DEFECTIVE BEARINGS (1) Anti friction	Early stages - 30K 60K cpm depending on size and speed Late stages - high 1x and multiple harmonics	radial except higher axial on thrust bearing	N/A	increases as bearing degrades, may disappear just before failure	Bandwidth broadens as bearing degrades. Baseline may increase across entire spectrum	$BPFO = N B/2 * cpm [1 - Bd/PD \cos \theta]$ $BPFI = N B/2 * cpm [1 + Bd/PD \cos \theta]$ $FTF = cpm/2 [1 - Bd/PD \cos \theta]$ $BSF = PD/2*Bd * cpm [1 - (Bd/PD)^2 \cos^2 \theta]$
(2) Sleeve	Early stages - sub-harmonics (may only be noticeable on shaft) Late stages - will appear as mechanical looseness (see below)	radial	Shaft proximity probe or bits will indicate shaft position and dynamic changes	increases as bearing degrades	high baseline energies below 1x, 2x and 3x rpm	Monitoring of rotor position (thrust) via proximity probes can provide reliable protection against thrust bearing failure

MECHANICAL LOOSENESS (1) Bearings, pedestals, etc (non-rotating)	1x, 2x and 3x predominant may be up to 10x at lower amplitude	radial	varies with type of looseness	steady	may extend up to 10 shaft speed	
(2) Impellers, etc (rotating)	1x predominant, but may have harmonics up to 10x at low levels	radial	will vary from start up to start-up	steady while running, but will vary from start up to start up	may extend up to 10x shaft speed	Variance of amplitude or phase may be caused by center of gravity shifts.
OPERATION (process-related) (1) Blade/vane pass	no. of blades/vane x rpm	radial predominant in the direction of discharge piping	N/A	fluctuating	broadband	More than one (1) discharge volute will produce harmonics of blade passing frequency.
(2) Cavitation or starvation	random - broadband	radial	N/A	fluctuating	broadband up to 2000 + Hz	Vane pass frequencies may be superimposed.
DRIVE BELTS (1) Mis-matched, worn or stretched - also applies to adjustable sheave applications	many multiples of belt frequency but 2x belt frequency usually dominant	radial, especially high in line with belts	N/A	may be unsteady and beating if a belt freq. is close to driver or driven speed	N/A	Belt frequency = $\frac{3.14 \text{ rpm} \times \text{pitch diameter}}{\text{belt length}}$
(2) Eccentric and/or unbalanced sheaves	1x shaft speed	radial	in-phase	steady	N/A	Balancing is possible with washers applied to taper lock bolts.
(3) Drive belt or sheave face misalignment	1x driver shaft	axial	in phase	steady	N/A	Confirm with a straight edge.
(4) Drive belt resonance	belt resonance with no relationship to rotational speeds	radial	N/A	may be unsteady	± 20 percent of resonant frequency depending upon damping	Confirm with strobe light and belt excitation techniques. Change belt tension or belt length to eliminate the problem.
RESONANCE	requires forcing function to excite its natural frequencies	axial or radial	A center hung rotor resonance will display 180° out-of-phase bearing relationships. A component within a structure will display phase relationships dependent upon the bending mode excited	steady, but baseline energy fluctuations depend on force and damping	appears broad at base, width depends upon damping	Frequency is independent of speed changes
INSTABILITY (1) Oil whirl	40 to 46 percent of running speed	radial	N/A	steady	discrete peaks	considered excessive when amplitudes reach 50% of normal bearing clearances
(2) Oil whip	Sub-rotational and equal to shaft resonance	radial	N/A	steady	discrete peaks	
(3) Rotor rub	50 percent of running speed and half harmonics	radial	N/A	steady	discrete peaks	Rub increases the resonant frequency to the next highest fraction of running speed
GEARS (1) Transmission error (poorly finished tooth face)	gear mesh frequency (gear rpm x no. of teeth) and harmonics	radial for spur gears, axial for helical or herringbone gears	N/A	depends on loading, speed and total transmission error	usually single peak, but sometimes low sidebands	
(2) Pitch line runout, mass unbalance, misalignment or faulty tooth	1x rpm and gear mesh frequency with ± gear rpm sidebands	radial for spur gears, axial for helical or herringbone gears	N/A	1x rpm and mesh frequency sidebands depend on fault severity	discrete peaks	May excite lateral or torsional resonances at various frequencies. Machining errors during hobbing can cause high 2x or 3x gear rpm vibration

NOTE 1 This guide is based on casing measurements unless otherwise noted.

NOTE 2 Vibration symptoms are based upon the observation of amplitude on a logarithmic scale.

Appendix B. Predicates and Code of Editor Program

Predicates will be discussed in a fashion similar to the Turbo Prolog 2.0 Reference Guide (10). This means that they will be listed by name, function performed, what domain types are used in the call to the predicate, which objects must be known and those that are determined by the predicate in normal operation and under what conditions does it fail. Two standard predicates are used. One is menu2, that presents a list and returns the list position of the item chosen on that list. The other is longmm, an adaptation of longmenu, that allows for list that can be scrolled and returns the line of the choice in the list chosen plus the key pressed to choose that choice.

Predicates - Editor System

act_part_2

Function — Acts upon the option chosen off of the rule menu for the rule RULENAME. Ends in run, part_2, or part_3.

```
Declaration    act_part_2 (fkey(INTEGER), RULENAME)
Domains        (KEY, STRING)
Flow patterns  (i,i)
Fails          Never
```

act_part_3

```
Function       Acts upon the option chosen on the edit rule
               part menu for the rule RULENAME.  Ends in
               cond_editor, mtd_editor, std_editor,
               ref_editor, or part_2.
Declaration    act_part_3 (fkey(INTEGER), RULENAME)
Domains        (KEY, STRING)
Flow pattern   (i,i)
Fails          If integer not 1, 2, 3, 4, or 5.
```

act_run

```
Function       Acts upon the option chosen off of the
               choose rule menu.  Ends in run, act_run
               (fkey(3)) ,part_2, or exit.
Declaration    act_run (fkey(INTEGER))
Domains        (KEY)
Flow patterns  (i)
Fails          Never
```

append

```
Function       Connects two list together into a third
               list.
Declaration    append (FIRSTLIST, SECONDLIST, COMBINEDLIST)
Domains        (STRINGLIST, STRINGLIST, STRINGLIST) or
               (MTDL, MTDL, MTDL) or
               (STANDARDL, STANDARDL, STANDARDL) or
               (PUBL, PUBL, PUBL)
Flow pattern   (i,i,o)
Fails          Never
```

append_change

```
Function       Changes the confidence factors for a
               condition in a list of conditions for a
               rule.
Declaration    append_change (CSTAR, MODIFIEDELEMENT,
               NEWCSTAR)
Domains        (CSTAR, CSTAR, CSTAR)
Flow pattern   (i,i,o)
```

Fails	Never

appendcstar

Function	Connects two list together into a third list.
Declaration	appendcstar (FIRSTLIST, SECONDLIST, COMBINEDLIST)
Domains	(CSTAR, CSTAR, CSTAR)
Flow pattern	(i,i,o)
Fails	Never

create_list

Function	Builds a list LIST of the names of all the rules in the current database.
Declaration	create_list ([], LIST)
Domains	(STRINGLIST, STRINGLIST)
Flow pattern	(i,o)
Fails	Never

cond_editor

Function	Allows the user to modify the conditions in the database and their use in rules. Ends in act_part_3 (fkey(1),_) or part_3.
Declaration	cond_editor ((3 or 4), RULENAME, CHOICE, fkey(1 to 4))
Domains	(INTEGER, STRING, INTEGER, KEY)
Flow pattern	(i,i,i,i)
Fails	Never

cut_cond

Function	Converts the condition list LISTofCOND for a rule into a list of conditions LIST belonging to the rule
Declaration	cut_cond (LISTofCOND, [], LIST)
Domains	(CSTAR, STRINGLIST, STRINGLIST)
Flow pattern	(i,i,o)
Fails	Never

cut_mtd

Function	Converts the suggested further action list SFA for a rule into a list of suggested further actions LIST belonging to the rule
Declaration	cut_mtd (SFA, [], LIST)
Domains	(MTDL, STRINGLIST, STRINGLIST)
Flow pattern	(i,i,o)
Fails	Never

cut_ref

Function	Converts the references belonging to a rule, REFERENCELIST, into a list LIST of references belonging to the rule
Declaration	cut_ref (REFERENCELIST, [], LIST)
Domains	(PUBL, STRINGLIST, STRINGLIST)
Flow pattern	(i,i,o)
Fails	Never

cut_std

Function	Converts the standards belonging to a rule, STANDARDSLIST, into a list LIST of standards belonging to the rule
Declaration	cut_std (STANDARDSLIST, [], LIST)
Domains	(STANDARDL, STRINGLIST, STRINGLIST)
Flow pattern	(i,i,o)
Fails	Never

editor

Function	Controls cursor movement between column 17 and Maximum_Column, and rows 1 through 9. Can write any characters and use the back delete key, along with left and right arrows at a skip rate of five positions with each press.
Declaration	editor (Row, Column, Maximum_Column, Key_Pressed)
Domains	(INTEGER, INTEGER, INTEGER, KEY)
Flow patterns	(i,i,i,i) The three integers
Fails	Key not defined

element

Function	Determines the element ELEMENT located NUMBERth in a list LIST
Declaration	element (NUMBER, LIST, ELEMENT)
Domains	(INTEGER, STRINGLIST, STRING)
Flow pattern	(i,i,o)
Fails	List of length less than NUMBER

elements

Function	Determines the element ELEMENT located NUMBERth in a list LIST
Declaration	elements (NUMBER, LIST, ELEMENT)
Domains	(INTEGER, CSTAR, CSTAR)
Flow pattern	(i,i,o)
Fails	List of length less than NUMBER

error

Function	In the event of a heap or stack overflow in memory during operation, it saves the database currently being worked on and returns the user to the choose database window.
Declaration	error (ERROR_NUMBER)
Domains	(INTEGER)
Flow pattern	(i)
Fails	Never

find_cond_value

Function	Returns the confidence factors for the NUMBERth condition in a condition list for a rule.
Declaration	find_cond_value (LIST, NUMBER,NO_FACTOR, MAYBE_FACTOR)
Domains	(CSTAR, INTEGER, REAL, REAL)
Flow pattern	(i,i,o,o)
Fails	LIST shorter in length than NUMBER

lastnumber_cond

Function	Determines the largest reference number for the conditions that are present in the database
Declaration	lastnumber_cond (1, LARGEST)

Domains	(INTEGER, INTEGER)
Flow pattern	(i,o)
Fails	Never

lastnumber_mtd

Function	Determines the largest reference number for the suggested further actions that are present in the database
Declaration	lastnumber_mtd (1, LARGEST)
Domains	(INTEGER, INTEGER)
Flow pattern	(i,o)
Fails	Never

lastnumber_ref

Function	Determines the largest reference number for the references that are present in the database
Declaration	lastnumber_ref (1, LARGEST)
Domains	(INTEGER, INTEGER)
Flow pattern	(i,o)
Fails	Never

lastnumber_rule

Function	Determines the largest reference number for the rules that are present in the database
Declaration	lastnumber_rule (1, LARGEST)
Domains	(INTEGER, INTEGER)
Flow pattern	(i,o)
Fails	Never

lastnumber_std

Function	Determines the largest reference number for the standards that are present in the database
Declaration	lastnumber_std (1, LARGEST)
Domains	(INTEGER, INTEGER)
Flow pattern	(i,o)
Fails	Never

make_new_cond_list

Function	Creates a new condition list NEWLIST from an old condition list OLDLIST less the NUMBERth element
Declaration	make_new_cond_list (OLDLIST, NUMBER, [], NEWLIST)
Domains	(CSTAR, INTEGER, CSTAR, CSTAR)
Flow pattern	(i,i,i,o)
Fails	Never

make_new_mtd_list

Function	Creates a new list NEWLIST of suggested actions for solved difficulties, from an old suggested actions list OLDLIST less the NUMBERth element.
Declaration	make_mtd_cond_list (OLDLIST, NUMBER, [], NEWLIST)
Domains	(MTDL, INTEGER, MTDL, MTDL)
Flow pattern	(i,i,i,o)
Fails	Never

make_new_ref_list

Function	Creates a new reference list NEWLIST from an old reference list OLDLIST less the NUMBERth element
Declaration	make_new_std_list (OLDLIST, NUMBER, [], NEWLIST)
Domains	(PUBL, INTEGER, PUBL, PUBL)
Flow pattern	(i,i,i,o)
Fails	Never

make_new_std_list

Function	Creates a new standard list NEWLIST from an old standard list OLDLIST less the NUMBERth element
Declaration	make_new_std_list (OLDLIST, NUMBER, [], NEWLIST)
Domains	(STANDARDL, INTEGER, STANDARDL, STANDARDL)
Flow pattern	(i,i,i,o)
Fails	Never

mtd_editor

Function	Allows the user to modify the suggested further actions that ought to be done in the case of a successful diagnosis. Ends in act_part_3 (fkey(2),_) or part_3.
Declaration	mtd_editor (fkey(INTEGER), RULENAME, CHOICE)
Domains	(KEY, STRING, INTEGER)
Flow pattern	(i,i,i)
Fails	Never

part_2

Function	Prints out the rule menu. Ends in act_part_2
Declaration	part_2 (RULENAME)
Domains	(STRING)
Flow patterns	(i)
Fails	Never

part_3

Function	Prints out the menu of what rule parts may be edited. Ends out in act_part_3.
Declaration	part_3 (RULENAME)
Domains	(STRING)
Flow pattern	(i)
Fails	Never

ref_editor

Function	Allows the user to modify the references in the database and related to rules. Ends in act_part_3 (fkey(3),_) or part_3.
Declaration	ref_editor (fkey(INTEGER), RULENAME, CHOICE)
Domains	(KEY, STRING, INTEGER)
Flow pattern	(i,i,i)
Fails	Never

run

Function	Prints out the choose rule menu. Ends in act_run.
Declaration	run
Domains	
Flow patterns	
Fails	Never

stall

Function	Places a window with the message "Please Wait, Reading Disk " printed in it.
Declaration	stall
Domains	
Flow pattern	
Fails	Never

std_editor

Function	Allows the user to modify the standards in the database and related to rules. Ends in act_part_3 (fkey(3),_) or part_3.
Declaration	std_editor (fkey(INTEGER), RULENAME, CHOICE)
Domains	(KEY, STRING, INTEGER)
Flow pattern	(i,i,i)
Fails	Never

test

Function	Checks if key pressed is either Y or y Declaration test (Response)
Domains	(CHAR)
Flow patterns	(i)
Fails	If character is not Y or y

title_page

Function	Prints out introductory information to the current window
Declaration	title_page
Domains	
Flow patterns	
Fails	Never

update_3_cond

Function	Makes list of conditions belonging to a rule RULENAME
Declaration	update_3_cond (RULENAME, LIST)
Domains	(STRING, STRINGLIST)
Flow pattern	(i,o)
Fails	RULENAME not in database

update_3_mtd

```
Function        Makes list of suggested repairs for actions
                belonging to a rule RULENAME
Declaration     update_3_mtd (RULENAME, LIST)
Domains         (STRING, STRINGLIST)
Flow pattern    (i,o)
Fails           RULENAME not in database
```

update_3_ref

```
Function        Makes list of references belonging to a rule
                RULENAME
Declaration     update_3_ref (RULENAME, LIST)
Domains         (STRING, STRINGLIST)
Flow pattern    (i,o)
Fails           RULENAME not in database
```

update_3_std

```
Function        Makes list of standards belonging to a rule
                RULENAME
Declaration     update_3_std (RULENAME, LIST)
Domains         (STRING, STRINGLIST)
Flow pattern    (i,o)
Fails           RULENAME not in database
```

update_4_cond

```
Function        Makes a list of all conditions in the
                database
Declaration     update_4_cond ([], LIST)
Domains         (STRINGLIST, STRINGLIST)
Flow pattern    (i,o)
Fails           Never
```

update_4_mtd

```
Function        Makes a list of all suggested actions in the
                database
Declaration     update_4_mtd ([], LIST)
Domains         (STRINGLIST, STRINGLIST)
Flow pattern    (i,o)
Fails           Never
```

update_4_ref

```
Function        Makes a list of all references in the
                database
Declaration     update_4_ref ([], LIST)
Domains         (STRINGLIST, STRINGLIST)
Flow pattern    (i,o)
Fails           Never
```

update_4_std

```
Function        Makes a list of all standards in the
                database
Declaration     update_4_std ([], LIST)
Domains         (STRINGLIST, STRINGLIST)
Flow pattern    (i,o)
Fails           Never
```

Code - Editor Program

```
code = 3000
include "menu2.pro"
include "longmm.pro"

DOMAINS

 db_selector = mydba
 dbdom = rule (RNO,REAL,CSTAR,STRING,PUBL,MTDL,STANDARDL);
      cond (CNO,STRING,STRING);
      library (INTEGER,STRING,STRING,STRING,STRING,STRING);
      more_to_do (MTDNO,STRING);
      standard (INTEGER,STRING,STRING,STRING,STRING,STRING,STRING)

 AUTHOR,TITLE,PUBLISH,PDATE,FIXES,SUMMARY,SOURCE = STRING
 RNO,CNO,PUB,MTDNO,STANDARD = INTEGER

 CSTAR = CONDITIONL*
 CONDITIONL = C*
 C = cn(CNO);r(REAL)
 RESPONCEL = INTEGER*
 PUBL = INTEGER*
 MTDL = MTDNO*
 RNOL = RNO*
 STANDARDL = INTEGER*
 file = file1; file2
 NUML = INTEGER*

DATABASE
 snake (STRING)

PREDICATES

 editor(INTEGER,INTEGER,INTEGER,KEY)
 test (CHAR)
 title_page
 run
 act_run (KEY)
 part_2 (STRING)
 act_part_2 (KEY,STRING)
 part_3 (STRING)
 act_part_3 (KEY,STRING)
 create_list (STRINGLIST,STRINGLIST)
 append (STRINGLIST,STRINGLIST,STRINGLIST)
 append (MTDL,MTDL,MTDL)
 append (STANDARDL,STANDARDL,STANDARDL)
 append (PUBL,PUBL,PUBL)
 appendcstar (CSTAR,CSTAR,CSTAR)
 append_change (CSTAR,CSTAR,CSTAR)
 element (INTEGER,STRINGLIST,STRING)
```

```
 elements (INTEGER,CSTAR,CSTAR)
 find_cond_value (CSTAR,INTEGER,REAL,REAL)
 update_3_cond (STRING,STRINGLIST)
 update_3_mtd (STRING,STRINGLIST)
 update_3_std (STRING,STRINGLIST)
 update_3_ref (STRING,STRINGLIST)
 update_4_cond (STRINGLIST,STRINGLIST)
 update_4_mtd (STRINGLIST,STRINGLIST)
 update_4_std (STRINGLIST,STRINGLIST)
 update_4_ref (STRINGLIST,STRINGLIST)
 cond_editor (INTEGER,STRING,INTEGER,KEY)
 mtd_editor (KEY,STRING,INTEGER)
 std_editor (KEY,STRING,INTEGER)
 ref_editor (KEY,STRING,INTEGER)
 cut_cond (CSTAR,STRINGLIST,STRINGLIST)
 cut_mtd (MTDL,STRINGLIST,STRINGLIST)
 cut_std (STANDARDL,STRINGLIST,STRINGLIST)
 cut_ref (PUBL,STRINGLIST,STRINGLIST)
 make_new_cond_list (CSTAR,INTEGER,CSTAR,CSTAR)
 make_new_mtd_list (MTDL,INTEGER,MTDL,MTDL)
 make_new_std_list (STANDARDL,INTEGER,STANDARDL,STANDARDL)
 make_new_ref_list (PUBL,INTEGER,PUBL,PUBL)
 lastnumber_rule (INTEGER,INTEGER)
 lastnumber_cond (INTEGER,INTEGER)
 lastnumber_mtd (INTEGER,INTEGER)
 lastnumber_std (INTEGER,INTEGER)
 lastnumber_ref (INTEGER,INTEGER)
 stall
 error (INTEGER)

GOAL
trace (off),
 makewindow (1,31,71,"",0,0,25,80),
 title_page,
 makewindow (1,32,1,"",0,0,16,80),
 write ("\n\n       Load an existing database by pressing (F5)  "),
 write ("\n\n                    or  "),
 write ("\n\n       Create a new database by pressing (F1).  "),
 makewindow (2,79,1,"",16,0,9,80),!,
 run.

CLAUSES
 run :-
  shiftwindow (2),
  clearwindow,
  write ("                Rule Menu       "),
  write ("\n_____________________________________________________________________"),
  write ("_____________"),
  write ("\n F1  Create a New DATABASE       "),
  write ("  F6  Exit the Editor program  "),
  write ("\n F2  Views all Rules in database   "),
  write ("\n F3  Add a New Rule to the database "),
```

```
  write ("\n  F4  Choose an Existing Rule to Edit "),
  write ("\n  F5  Choose a database to edit     "),
 repeat,
  readkey (KEY),
  trap (act_run(KEY),NUM,error(NUM)).
/*-----------------------------------------------------------------*/
act_run (fkey(1)):-
 db_close (mydba),
 shiftwindow (2),
 clearwindow,
 write ("\n\n   Enter the NAME, without extension, for the new database\n   "),
 write ("\n This will be the database which you will be working in unless"),
 write ("\n you choose to change databases by loading a new one or creating "),
 write ("\n a new database.\n   "),
 readln (NAME),
 concat (NAME,".dba",DosFileName),
 db_create (mydba,DosFileName,in_file),
 clearwindow,
 shiftwindow (1),
 window_attr (31),
 shiftwindow (2),
 run.

act_run (fkey(2)):-
 shiftwindow (1),
 scr_attr (5,70,N),
 not (N = 32),
 create_list ([],LIST),
 not (LIST = []),
 shiftwindow (2),
 clearwindow,
 write ("\n  Press any Function key to exit "),
 shiftwindow (1),
 ROW = 1,
 COLUMN = 1,
 LENGTH = 14,
 longmm(ROW,COLUMN,LENGTH,31,0,LIST,"",1,_,_),!,
 clearwindow,
 shiftwindow (2),
 run.

act_run (fkey(2)):-
 scr_attr (5,70,N),
 not (N = 32),
 shiftwindow (2),
 clearwindow,
 write ("\n\n   No rules in Database.  Transfering over to the CREATE\n",
      "  RULES option so that you may create some."),
 act_run (fkey(3)).

act_run (fkey(3)):-
 shiftwindow (1),
```

```
 scr_attr (5,70,N),
 not (N = 32),
 shiftwindow (2),
 clearwindow,
 write ("\n  Enter the reason of difficulty for this rule "),
 write ("\n  Keep the response to one line.  If you press Enter by accident,"),
 write ("\n  then use backspace to return to the first line and del to "),
 write ("\n  remove the second line from the rule's name.  = = > F10 to EXIT"),
 write ("\n            (Press F10 to exit)"),
 makewindow (3,31,0,"",22,5,2,70),
repeat,
 edit ("",STRING),
 removewindow,
 lastnumber_rule (1,NUMBER),
 RNO = NUMBER + 1,
 write ("\n\n  What is the maximum confidence in this rule ?\n    "),
 readreal (REAL),
 chain_insertz (mydba,rul,dbdom,rule(RNO,REAL,[[]],STRING,[],[],[]),_),
 write ("\n\n  The rule ",STRING," has been inserted into the database"),
 run.

act_run (fkey(4)):-
 shiftwindow (1),
 scr_attr (5,70,N),
 not (N = 32),
 create_list ([],LIST),
 not (LIST = []),
 shiftwindow (2),
 clearwindow,
 write ("\n  Place the lightbar over the desired rule and press any"),
 write ("\n     function key to pick up a rule to edit"),
 shiftwindow (1),
 ROW = 1,
 COLUMN = 1,
 LENGTH = 14,
 longmm(ROW,COLUMN,LENGTH,31,0,LIST,"",1,CHOICE,_),!,
 element (CHOICE,LIST,RULENAME),
 clearwindow,
 write ("  The Rule Chosen is : ",RULENAME),
 shiftwindow (2),
 clearwindow,!,
 part_2 (RULENAME).

act_run (fkey(4)):-
 scr_attr (5,70,N),
 not (N = 32),
 shiftwindow (2),
 clearwindow,
 write ("\n\n   No rules in Database.  Transfering over to the CREATE\n",
     " RULES option so that you may create some."),
 act_run (fkey(3)).
```

```
act_run (fkey(5)):-
 db_close (mydba),
 shiftwindow (1),
 clearwindow,
 write ("\n          Choose the data base with which you want to work"),
 write ("\n          Press F4 if you need to change the drive sourse"),
 makewindow (3,7,64,"",5,10,8,60),
 dir ("","*.dba",DosFileName,1,1,1),
 db_open (mydba,DosFileName,in_file),
 removewindow,
 clearwindow,
 window_attr (31),
 run.

act_run (fkey(6)):-
  makewindow (1,15,70,"y or n",8,33,3,17),
  write ("Are You Sure ?"),
  readchar (A),
  removewindow,
  test (A),!,
  db_close (mydba),
  clearwindow,
 removewindow,
 removewindow,
 removewindow,
 text,
  exit.

act_run (_) :-
  sound (75,75),
  fail.
/*-----------------------------------------------------------------*/
 create_list ([],_):-
  chain_terms (mydba,rul,dbdom,rule(_,_,_,STRING,_,_,_),_),
  assertz (snake (STRING)),
  fail.

 create_list (LIST,LISTA):-
  retract (snake (FIRST)),
  append (LIST,[FIRST],LISTB),
  create_list (LISTB,LISTA).

 create_list (LIST,LISTA):-!,
  LISTA = LIST.
/*-----------------------------------------------------------------*/
 part_2 (RULENAME):-
  clearwindow,
  write ("  What would you like to do to this Rule ?"),
  write ("\n_____________________________________________________________________"),
  write ("_____________"),
  write ("\n   F1    Delete the Rule From the Database "),
  write ("\n   F2    Modify the Rule's Conditons and Related Information "),
```

```
 write ("\n   F3    Edit the Rule's Name and Confidence (OCF)"),
 write ("\n   F6    Exit to the Previous Menu, (Database Menu) "),
repeat,
 readkey (KEY),
 act_part_2 (KEY,RULENAME).
/*-------------------------------------------------------------------*/
 act_part_2 (fkey(1),RULENAME) :-
  clearwindow,
  write ("\n\n  Are You Sure You want to Delete the Rule\n :",RULENAME," ?"),
  ROW = 14,
  COLUMN = 35,
  menu (ROW,COLUMN,7,64,["  Yes ","  No  "],"",1,CHOICE),
  CHOICE = 1,!,
  shiftwindow (1),
  clearwindow,
  write ("\n  The rule ",RULENAME," has been deleted from the database"),
  chain_terms (mydba,rul,dbdom,rule(_,_,_,RULENAME,_,_,_),REF),
  term_delete (mydba,rul,REF),
  shiftwindow (2),
  clearwindow,
  run.

 act_part_2 (fkey(1),RULENAME) :-!,
  clearwindow,
  write ("\n\n   Entry in Rule Database not Deleted  "),
  write ("\n\n\n   (Press Any Key to Continue)   "),
  readkey (_),
  part_2 (RULENAME).

 act_part_2 (fkey(2),RULENAME) :-!,
  part_3 (RULENAME).

 act_part_2 (fkey(3),RULENAME) :-
  chain_terms (mydba,rul,dbdom,rule(A,CONFIDENCE,C,RULENAME,E,F,G),REF),
  clearwindow,
  shiftwindow(2),
  write ("\n Edit the Rule's reason.  Press F10 when correct"),
  write ("\n Make sure to keep response to one line.  If you by accident"),
  write ("\n press the Enter key, use Backspace followed by Delete to"),
  write ("\n remove the second line."),
  makewindow (4,7,64,"Rule's Reason",14,0,4,80),
  edit (RULENAME,STRING),
  removewindow,
  clearwindow,
  shiftwindow (1),
  clearwindow,
  write ("\n\n   The Maximum confidence in the rule ",RULENAME," was ",CONFIDENCE,",",
      "\n\n   What Maximum confidence do you have in ",STRING," ?"),
  write ("\n   "),
  repeat,
  readreal (CON),
  term_replace (mydba,dbdom,REF,rule(A,CON,C,STRING,E,F,G)),
```

```
  write ("\n  The rule is now        : ",STRING,
      "\n  Its Maximum Confidence is : ",CON),
  write ("\n\n\n\n     (Press Any Key To Continue)"),
  readkey(_),
  clearwindow,
  write (" Rule : ",STRING),
  shiftwindow (2),
  part_2 (STRING).

 act_part_2 (fkey(6),_) :-!,
  shiftwindow(1),
  clearwindow,
  shiftwindow(2),
  run.

 act_part_2 (_,_) :-
  fail.
/*-------------------------------------------------------------------*/
part_3 (RULENAME) :-
 shiftwindow (2),
 clearwindow,
  write (" What part of this Rule would you like to Edit ?"),
  write ("\n_____________________________________________________________"),
  write ("\n  F1   Conditions "),
  write ("\n  F2   Suggestions of Corrective Procedures "),
  write ("\n  F3   Related Standards "),
  write ("\n  F4   References "),
  write ("\n  F6   Exit to previous menu  (Rule menu)"),
  repeat,
  readkey (KEY),
  act_part_3 (KEY,RULENAME).
/*-------------------------------------------------------------------*/

act_part_3 (fkey(1),RULENAME) :-
 stall,
 update_3_cond (RULENAME,LIST),
 removewindow,
 not (LIST = []),!,
 shiftwindow(1),
 clearwindow,
 write (" Rule : ",RULENAME),
 shiftwindow (2),
 clearwindow,
 write (" What would you like to do to the Conditions ?"),
 write ("\n_____________________________________________________________"),
 write ("\n  F1   Modify a Condition's Confidence Factors "),
 write ("\n  F2   Remove a Condition From the Rule "),
 write ("\n  F3   Add or Create a Condition to/for the Rule"),!,
 write ("\n  F6   Exit to previous menu  (Parts menu)"),
 ROW = 2,
 COLUMN = 1,
 LENGTH = 15,
```

```
 longmm(ROW,COLUMN,LENGTH,31,0,LIST,"",1,CHOICE,KEY),!,
 cond_editor (3,RULENAME,CHOICE,KEY).

act_part_3 (fkey(1),RULENAME) :-
 clearwindow,
 write ("\n\n\n     No Conditions Assigned to this Rule.",
     "\n\n\n   Transfering over to the Conditions screen"),
 sound (100,250),
 cond_editor (3,RULENAME,0,fkey(3)).
/*-----------------------------------------------------------------*/
act_part_3 (fkey(2),RULENAME) :-
 shiftwindow (1),
 clearwindow,
 write (" Rule : ",RULENAME),
 stall,
 update_3_mtd (RULENAME,LIST),
 removewindow,
 not (LIST = []),!,
 shiftwindow(2),
 clearwindow,
 write ("  What would you like to do to the Corrective Procedures"),
 write ("\n_____________________________________________________________________"),
 write ("\n   F1    Remove a Suggested Corrective Procedure from the rule"),
 write ("\n   F2    Add a Suggested Corrective Procedure to the rule"),
 write ("\n   F3    Modify a Corrective Procedure in the database"),
 write ("\n   F4    Create a New Corrective Procedure for the database"),
 write ("\n   F6    Return to Previous Menu  (Parts Menu) "),
 ROW = 2,
 COLUMN = 1,
 LENGTH = 15,
 longmm(ROW,COLUMN,LENGTH,31,0,LIST,"",1,CHOICE,KEY),
 mtd_editor (KEY,RULENAME,CHOICE).

act_part_3 (fkey(2),RULENAME) :-
 clearwindow,
 write ("\n\n\n  No Corrective Procedures Assigned to this Rule.  Transfering",
     "\n\n\n  Over to the Add Suggested Corrective Procedures screen"),
 sound (100,250),
 mtd_editor (fkey(2),RULENAME,0).
/*-----------------------------------------------------------------*/
act_part_3 (fkey(3),RULENAME) :-
 shiftwindow (1),
 clearwindow,
 write (" Rule : ",RULENAME),
 stall,
 update_3_std (RULENAME,LIST),
 removewindow,
 not (LIST = []),!,
 shiftwindow(2),
 clearwindow,
 write ("  What would you like to do to the Standards"),
 write ("\n_____________________________________________________________________"),
```

```
 write ("\n   F1    Remove a Standard from the Rule "),
 write ("\n   F2    Add a Standard to the Rule "),
 write ("\n   F3    Modify a current Standard in the database"),
 write ("\n   F4    Create a New Standard for the database"),
 write ("\n   F6    Return to Previous Menu  (Parts menu)"),
 ROW = 2,
 COLUMN = 1,
 LENGTH = 15,!,
 longmm(ROW,COLUMN,LENGTH,31,0,LIST,"",1,CHOICE,KEY),!,
 std_editor (KEY,RULENAME,CHOICE).

act_part_3 (fkey(3),RULENAME) :-
 clearwindow,
 write ("\n\n\n   No Standards are assigned to this Rule.  Transfering",
      "\n\n\n   Over to the Add Standards screen"),
 sound (100,250),
 std_editor (fkey(2),RULENAME,0).
/*-----------------------------------------------------------------*/
act_part_3 (fkey(4),RULENAME) :-
 shiftwindow (1),
 clearwindow,
 write (" Rule : ",RULENAME),
 stall,
 update_3_ref (RULENAME,LIST),
 removewindow,
 not (LIST = []),!,
 shiftwindow(2),
 clearwindow,
 write ("  What would you like to do to the References"),
 write ("\n_____________________________________________________________________________"),
 write ("\n   F1    Remove a Reference from the rule"),
 write ("\n   F2    Add a Reference to the rule"),
 write ("\n   F3    Modify a current Reference in the database"),
 write ("\n   F4    Create a New Reference for the database"),
 write ("\n   F6    Return to Previous Menu  (Parts menu)"),
 ROW = 2,
 COLUMN = 1,
 LENGTH = 15,
 longmm(ROW,COLUMN,LENGTH,31,0,LIST,"",1,CHOICE,KEY),!,
 ref_editor (KEY,RULENAME,CHOICE).

act_part_3 (fkey(4),RULENAME) :-
 clearwindow,
 write ("\n\n\n   No References are assigned to this Rule.  Transfering",
      "\n\n\n   over to the Add References screen"),
 sound (100,250),
 ref_editor (fkey(2),RULENAME,0).
/*-----------------------------------------------------------------*/
act_part_3 (fkey(6),RULENAME) :-
 shiftwindow (2),
 part_2 (RULENAME).
/*-----------------------------------------------------------------*/
```

```
act_part_3 (_,_) :-
 beep,
 beep,
 fail.
/*-----------------------------------------------------------------*/
cond_editor (3,RULENAME,CHOICE,fkey(1)):-
 chain_terms(mydba,rul,dbdom,rule(A,B,CSTAR,RULENAME,C,D,E),REF),
 elements (CHOICE,CSTAR,[[cn(NUMBER),_,_]]),
 shiftwindow (1),
 chain_terms (mydba,con,dbdom,cond(NUMBER,STRING1,STRING2),_),
 write ("\n\n  The rule is ",RULENAME,
      "\n\n  The condition is\n",STRING1,
      "\n\n  The condition's suggested action is\n  ",STRING2),
 shiftwindow (2),
 find_cond_value (CSTAR,NUMBER,LCFN,LCFM),
 clearwindow,
 write ("\n The Condition's old CN for a NO answer was  ",LCFN,
      "\n The Condition's old CM for a DON'T KNOW was ",LCFM,
      "\n\n Enter the value for the condition's new CN for a NO answer\n"),
 repeat,
 readreal (LCF_NO),
 write ("\n Enter the value for the condition's new CM for a DON'T KNOW answer\n"),
repeat,
 readreal (LCF_MAYBE),
 NEWELEMENT = [[cn(NUMBER),r(LCF_NO),r(LCF_MAYBE)]],
 append_change (CSTAR,NEWELEMENT,NEWLIST),
 term_replace (mydba,dbdom,REF,rule(A,B,NEWLIST,RULENAME,C,D,E)),
 act_part_3 (fkey(1),RULENAME).

cond_editor (3,RULENAME,CHOICE,fkey(2)):-
 not (CHOICE = 0),
 stall,
 update_3_cond (RULENAME,LIST),
 removewindow,
 element (CHOICE,LIST,STRING1),
 shiftwindow (2),
 clearwindow,
 write ("\n  Do You Want to Remove the Condition\n",STRING1),
 menu (5,25,23,1,["  Yes ","  No  "],"",1,CHOOSE),
 CHOOSE = 1,!,
 chain_terms(mydba,rul,dbdom,rule(A,B,CSTAR,RULENAME,E,F,G),REF),
 make_new_cond_list (CSTAR,CHOICE,[],LIST1),
 term_replace (mydba,dbdom,REF,rule(A,B,LIST1,RULENAME,E,F,G)),
 act_part_3 (fkey(1),RULENAME).

cond_editor (3,RULENAME,_,fkey(2)):-
 act_part_3 (fkey(1),RULENAME).

cond_editor (3,RULENAME,_,fkey(3)):-
 stall,
 update_4_cond ([],LIST),
 removewindow,
```

```
 not (LIST = []),!,
 shiftwindow (2),
 clearwindow,
 write (" What would you like to do to the Available Conditions ?"),
 write ("\n_____"),
 write ("\n   F1    Choose a Condition to Add to the Rule "),
 write ("\n   F2    Modify an existing Condition in the database"),
 write ("\n   F3    Create a New Condition for the database"),
 write ("\n   F6    View the Rule's Current List of Conditions  (Conditions menu)"),
 shiftwindow (1),
 clearwindow,
stall,
 update_4_cond ([],LIST),
removewindow,
 write (" Rule : ",RULENAME),
 ROW = 2,
 COLUMN = 1,
 LENGTH = 14,
 longmm(ROW,COLUMN,LENGTH,7,0,LIST,"",1,CHOICE,KEY),!,
 cond_editor (4,RULENAME,CHOICE,KEY).

cond_editor (3,RULENAME,_,fkey(3)):-
 shiftwindow (2),
 clearwindow,
 beep,
 write ("\n\n   No available Conditions in the Database.  Moving over\n",
     " to the CREATE option so that ones can be created for the database."),
 beep,
 cond_editor (4,RULENAME,1,fkey(3)).

cond_editor (3,RULENAME,_,fkey(6)):-!,
 part_3 (RULENAME).

cond_editor (4,RULENAME,CHOICE,fkey(1)):-
 stall,
 update_4_cond ([],LIST),
 removewindow,
 element (CHOICE,LIST,STRING),
 chain_terms(mydba,con,dbdom,cond(CNO,STRING,_),_),
 shiftwindow (2),
repeat,
 clearwindow,
 write ("\nWhat loss of confidence factor would you like for a NO response ?\n"),
 readreal (CN),
 clearwindow,
 write ("\nWhat loss of confidence factor would you like for a DON'T KNOW response ?\n"),
 readreal (CM),
 clearwindow,
 write (" LCF NO       = ",CN,"\n LCF DON'T KNOW = ",CM),
 write ("\n\n  Are these the values you want ?"),
 menu (5,25,23,1,["  Yes ","  No  "],"",1,CHOOSE),
 CHOOSE = 1,
```

```
 chain_terms(mydba,rul,dbdom,rule(A,B,CSTAR,RULENAME,E,F,G),REF),
 STRING1 = [[cn(CNO),r(CN),r(CM)]],
 appendcstar (CSTAR,STRING1,NEWCSTAR),
 term_replace (mydba,dbdom,REF,rule(A,B,NEWCSTAR,RULENAME,E,F,G)),!,
 act_part_3 (fkey(1),RULENAME).


cond_editor (4,RULENAME,CHOICE,fkey(2)):-!,
 stall,
 update_4_cond ([],LIST),
 removewindow,
 element (CHOICE,LIST,STRING1),
 shiftwindow (2),
 clearwindow,
 write ("\n Edit the current section.  Press F10 to go to the next section"),
 write ("\n Keep responses to one line.  Use delete to eliminate extra "),
 write ("\n characters and lines if you by accident use the Enter key."),
 chain_terms (mydba,con,dbdom,cond(NO,STRING1,STRING2),REF),
 makewindow (4,7,64,"Condition Explanation",14,0,4,80),
 write (STRING2),
 makewindow (3,7,64,"Condition Description",5,0,4,80),
 edit (STRING1,STRING3),
 write (STRING3),
 shiftwindow (4),
 edit (STRING2,STRING4),
 removewindow (3,0),
 removewindow (4,0),
 term_replace(mydba,dbdom,REF,cond(NO,STRING3,STRING4)),!,
 act_part_3 (fkey(1),RULENAME).

cond_editor (4,RULENAME,_,fkey(3)):-!,
 shiftwindow (2),
 clearwindow,
 write ("\n Create the current section.  Press F10 to go to the next section"),
 write ("\n Keep answers to one line.  If you by accident press Return, then"),
 write ("\n press Backspace followed by Delete to remove the added line."),
 makewindow (4,7,64,"Action for Don'T Know",14,0,4,80),
 makewindow (3,7,64,"Condition Description",5,0,4,80),
 edit ("",STRING1),
 shiftwindow (4),
 edit ("",STRING2),
 removewindow (3,0),
 removewindow (4,0),
 stall,
 lastnumber_cond (1,NO),
 NUMBER = NO + 1,
 chain_insertz (mydba,con,dbdom,cond(NUMBER,STRING1,STRING2),_),!,
 removewindow (9,1),
 cond_editor (3,RULENAME,0,fkey(3)).

cond_editor (4,RULENAME,_,fkey(6)):-!,
 act_part_3 (fkey(1),RULENAME).
```

```
cond_editor (N,RULENAME,A,_):-!,
 beep,
 readkey(KEY),
 cond_editor (N,RULENAME,A,KEY).
/*-----------------------------------------------------------------*/
 mtd_editor (fkey(1),RULENAME,CHOICE):-
 stall,
  update_3_mtd (RULENAME,LIST),
 removewindow,
  element (CHOICE,LIST,STRING1),
  shiftwindow (2),
  clearwindow,
  write ("\n  Do you want to remove the Suggested Corrective Procedures of\n",STRING1),
  menu (5,25,23,1,["  Yes ","  No  "],"",1,CHOOSE),
  CHOOSE = 1,!,
  chain_terms(mydba,mtd,dbdom,more_to_do(NUM,STRING1),_),
  chain_terms(mydba,rul,dbdom,rule(A,B,C,RULENAME,E,MTDL,G),REF),
  make_new_mtd_list (MTDL,NUM,[],MTD),
  term_replace (mydba,dbdom,REF,rule(A,B,C,RULENAME,E,MTD,G)),
  act_part_3 (fkey(2),RULENAME).

 mtd_editor (fkey(1),RULENAME,_):-!,
  clearwindow,
  write ("\n\n   Suggested Corrective Procedure not removed from Rule  "),
  write ("\n\n\n   (Press Any Key to Continue)  "),
  readkey (_),
  act_part_3 (fkey(2),RULENAME).

 mtd_editor (fkey(2),RULENAME,_):-
 stall,
  update_4_mtd ([],LIST),
 removewindow,
  not (LIST = []),!,
  shiftwindow (2),
  clearwindow,
  write ("\n\n  Place the light bar over the Suggested Procedure that you "),
  write ("\n  want to Add to the Rule.  Press any function key to Add it."),
  shiftwindow (1),
  clearwindow,
  write ("These are the Suggested Procedures Currently Stored in the Database"),
  ROW = 2,
  COLUMN = 1,
  LENGTH = 14,
  longmm (ROW,COLUMN,LENGTH,7,0,LIST,"",1,CHOICE,_),

  element (CHOICE,LIST,STRING),
  stall,
  chain_terms (mydba,mtd,dbdom,more_to_do(NUM,STRING),_),
  chain_terms (mydba,rul,dbdom,rule(A,B,C,RULENAME,E,MTDL,G),REF),
  append (MTDL,[NUM],MTD),
  term_replace (mydba,dbdom,REF,rule(A,B,C,RULENAME,E,MTD,G)),
```

```
 removewindow,
 act_part_3 (fkey(2),RULENAME).

mtd_editor (fkey(2),RULENAME,_):-!,
 shiftwindow (2),
 clearwindow,
 beep,
 write ("\n\n   No Suggested Corrective Procedures in Database. Moving over\n",
     " to the CREATE option so that ones can be created for the database."),
 beep,
 mtd_editor (fkey(4),RULENAME,1).

mtd_editor (fkey(3),RULENAME,_):-!,
stall,
 update_4_mtd ([],LIST),
removewindow,
 clearwindow,
 shiftwindow (1),
 clearwindow,
 write ("  RULE : ",RULENAME),
 ROW = 2,
 COLUMN = 1,
 LENGTH = 14,
 longmm (ROW,COLUMN,LENGTH,7,0,LIST,"",1,CHOICE,_),
 element (CHOICE,LIST,STRING),
 chain_terms (mydba,mtd,dbdom,more_to_do(NUM,STRING),REF),
 shiftwindow (2),
 clearwindow,
 write ("\n  The Current Suggested Corrective Procedure is\n       ",STRING,
     "\n\n   Modify the SCP to the desired form and then press",
     "\n  F10 to once the desired changes are made."),
 makewindow (3,7,64,"Suggested Corrective Procedure",12,0,4,80),
 edit (STRING,STRING1),
 removewindow (3,0),
 term_replace (mydba,dbdom,REF,more_to_do(NUM,STRING1)),
 act_part_3 (fkey(2),RULENAME).

mtd_editor (fkey(4),RULENAME,_):-!,
  clearwindow,
  shiftwindow (2),
  clearwindow,
  write ("\n Write in the Suggested Corrective Procedure.  Press F10 when it's"),
  write ("\n correct.  Remember to keep responses to one line.  If you"),
  write ("\n accidentaly press the Enter key, pressing Backspace followed"),
  write ("\n by Delete will remove the added line."),
  makewindow (4,7,64,"",12,0,4,80),
  edit ("",STRING),
  removewindow,
  lastnumber_mtd (1,NO),
  NUMBER = NO + 1,
  chain_insertz (mydba,mtd,dbdom,more_to_do(NUMBER,STRING),_),
  act_part_3 (fkey(2),RULENAME).
```

```
mtd_editor (fkey(6),RULENAME,_):-!,
 part_3 (RULENAME).

mtd_editor (_,RULENAME,A):-!,
 beep,
 readkey(KEY),
 mtd_editor (KEY,RULENAME,A).
/*-----------------------------------------------------------------*/
std_editor (fkey(1),RULENAME,CHOICE):-
stall,
 update_3_std (RULENAME,LIST),
removewindow,
 element (CHOICE,LIST,STRING),
 shiftwindow (2),
 clearwindow,
 write ("\n Do you want to remove the Standard \n",STRING),
 menu (5,25,23,1,["  Yes ","  No  "],"",1,CHOOSE),
 CHOOSE = 1,!,
 chain_terms(mydba,std,dbdom,standard(NUM,STRING,_,_,_,_,_),_),
 chain_terms(mydba,rul,dbdom,rule(A,B,C,RULENAME,E,F,STANDARDL),REF),
 make_new_std_list (STANDARDL,NUM,[],STANDARD),
 term_replace (mydba,dbdom,REF,rule(A,B,C,RULENAME,E,F,STANDARD)),
 act_part_3 (fkey(3),RULENAME).

std_editor (fkey(1),RULENAME,_):-!,
 clearwindow,
 write ("\n\n   Standard not removed from Rule  "),
 write ("\n\n\n   (Press Any Key to Continue)  "),
 readkey (_),!,
 act_part_3 (fkey(3),RULENAME).

std_editor (fkey(2),RULENAME,_):-
stall,
 update_4_std ([],LIST),
removewindow,
 not (LIST = []),!,
 shiftwindow (2),
 clearwindow,
 write ("\n Place the light bar over the Standard that you want to add to"),
 write ("\n the rule. Press any function key when you're ready to continue"),
 shiftwindow (1),
 clearwindow,
 write ("These Standards are the available ones in the Database"),
 ROW = 2,
 COLUMN = 1,
 LENGTH = 14,
 longmm (ROW,COLUMN,LENGTH,7,0,LIST,"",1,CHOICE,_),
 element (CHOICE,LIST,STRING),
 chain_terms (mydba,std,dbdom,standard(NUM,STRING,_,_,_,_,_),_),
 chain_terms (mydba,rul,dbdom,rule(A,B,C,RULENAME,E,F,STANDARDL),REF),
 append (STANDARDL,[NUM],STANDARD),
```

```
  term_replace (mydba,dbdom,REF,rule(A,B,C,RULENAME,E,F,STANDARD)),!,
  act_part_3 (fkey(3),RULENAME).

std_editor (fkey(2),RULENAME,_):-!,
  shiftwindow (2),
  clearwindow,
  beep,
  write ("\n\n   No available Standards in the Database.  Moving over\n",
       " to the CREATE option so that ones can be created for the database."),
  beep,
  std_editor (fkey(4),RULENAME,1).

std_editor (fkey(3),RULENAME,_):-!,
 stall,
  update_4_std ([],LIST),
removewindow,
  shiftwindow (2),
  clearwindow,
  write ("\n   Place the light bar over the Standard that you want to modify."),
  write ("\n Press any function key when you're ready to continue."),
  shiftwindow (1),
  clearwindow,
  write ("  Rule : ",RULENAME),
  ROW = 2,
  COLUMN = 1,
  LENGTH = 14,
  longmm (ROW,COLUMN,LENGTH,7,0,LIST,"",1,CHOICE,_),
  element (CHOICE,LIST,STRING),
  chain_terms (mydba,std,dbdom,standard(A1,STRING,C1,D1,E1,F1,G1),REF),
  shiftwindow (2),
  clearwindow,
  write ("\n\n  Use the arrow keys to move from line to line to edit text.",
       "  Use back delete\n to move the cursor left. Press esc to end",
       " editing."),
  shiftwindow (1),
  write ("\n Title       : ",STRING,
       "\n Source      : ",C1,
       "\n Date        : ",D1,
       "\n Summary     : ",E1,
       "\n              ",F1,
       "\n              ",G1),
  cursor (1,17),
  editor (1,17,76,left),!,
  field_str (1,17,60,B2),
  field_str (2,17,60,C2),
  field_str (3,17,60,D2),
  field_str (4,17,60,E2),
  field_str (5,17,60,F2),
  field_str (6,17,60,G2),
  term_replace (mydba,dbdom,REF,standard(A1,B2,C2,D2,E2,F2,G2)),
  shiftwindow (2),
  act_part_3 (fkey(3),RULENAME).
```

```
std_editor (fkey(4),RULENAME,_):-!,
 shiftwindow (2),
 clearwindow,
 write ("\n\n  Use the arrow keys to move from line to line to edit text.",
      "  Use back delete \n to move the cursor left.  Press esc to end",
      " editing."),
 shiftwindow (1),
 clearwindow,
 write ("  RULE : ",RULENAME),
 write ("\n Title       : ",
      "\n Source      : ",
      "\n Date        : ",
      "\n Summary     : ",
      "\n             ",
      "\n             "),
 cursor (1,17),
 editor (1,17,76,left),!,
 field_str (1,17,60,B2),
 field_str (2,17,60,C2),
 field_str (3,17,60,D2),
 field_str (4,17,60,E2),
 field_str (5,17,60,F2),
 field_str (6,17,60,G2),
 write ("\n\n  DO YOU WANT TO KEEP THIS STANDARD ? "),
 menu (15,25,23,1,["  Yes ","  No  "],"",1,CHOOSE),
 CHOOSE = 1,!,
 lastnumber_std (1,NO),
 NUM = NO + 1,
 chain_insertz (mydba,std,dbdom,standard(NUM,B2,C2,D2,E2,F2,G2),_),
 shiftwindow (2),
 act_part_3 (fkey(3),RULENAME).

std_editor (fkey(4),RULENAME,_):-!,
 act_part_3 (fkey(3),RULENAME).

std_editor (fkey(6),RULENAME,_):-!,
  part_3 (RULENAME).

std_editor (_,RULENAME,A):-!,
 beep,
 readkey(KEY),!,
 std_editor (KEY,RULENAME,A).
/*-----------------------------------------------------------------*/
 ref_editor (fkey(1),RULENAME,CHOICE):-
 stall,
  update_3_ref (RULENAME,LIST),
 removewindow,
  element (CHOICE,LIST,STRING),
  shiftwindow (2),
  clearwindow,
  write ("\n  Do you want to remove from the rule the Reference\n",STRING),
```

```
 menu (5,25,23,1,[" Yes "," No "],"",1,CHOOSE),
 CHOOSE = 1,!,
 chain_terms(mydba,lib,dbdom,library(NUM,_,STRING,_,_,_),_),
 chain_terms(mydba,rul,dbdom,rule(A,B,C,RULENAME,PUBL,F,G),REF),
 make_new_ref_list (PUBL,NUM,[],PUB),
 term_replace (mydba,dbdom,REF,rule(A,B,C,RULENAME,PUB,F,G)),
 act_part_3 (fkey(4),RULENAME).

ref_editor (fkey(1),RULENAME,_):-!,
 clearwindow,
 write ("\n\n   Reference not removed from Rule "),
 write ("\n\n\n   (Press Any Key to Continue)  "),
 readkey (_),!,
 act_part_3 (fkey(4),RULENAME).

ref_editor (fkey(2),RULENAME,_):-
stall,
 update_4_ref ([],LIST),
removewindow,
 not (LIST = []),!,
 shiftwindow (2),
 clearwindow,
 write ("\n  Place the light bar over the reference you would like to add"),
 write ("\n to the rule.  Press any function key to continue."),
 shiftwindow (1),
 clearwindow,
 write (" These are the References currently available in the database"),
 ROW = 2,
 COLUMN = 1,
 LENGTH = 14,
 longmm (ROW,COLUMN,LENGTH,7,0,LIST,"",1,CHOICE,_),
 element (CHOICE,LIST,STRING),
 stall,
 chain_terms (mydba,lib,dbdom,library(NUM,_,STRING,_,_,_),_),
 chain_terms (mydba,rul,dbdom,rule(A,B,C,RULENAME,PUBL,F,G),REF),
 append (PUBL,[NUM],PUB),
 term_replace (mydba,dbdom,REF,rule(A,B,C,RULENAME,PUB,F,G)),!,
 removewindow,
 act_part_3 (fkey(4),RULENAME).

ref_editor (fkey(2),RULENAME,_):-!,
 shiftwindow (2),
 clearwindow,
 beep,
 write ("\n\n   No available References in the Database.  Moving over\n",
      " to the CREATE option so that ones can be created for this rule."),
 beep,!,
 ref_editor (fkey(4),RULENAME,1).

ref_editor (fkey(3),RULENAME,_):-!,
 stall,
 update_4_ref ([],LIST),
```

```
removewindow,
 shiftwindow (2),
 clearwindow,
 write ("\n  Place the light bar over the Reference you want to edit. Press"),
 write ("\n any function key when you are ready to continue."),
 shiftwindow (1),
 clearwindow,
 write ("  Rule : ",RULENAME),
 ROW = 2,
 COLUMN = 1,
 LENGTH = 14,
 longmm (ROW,COLUMN,LENGTH,7,0,LIST,"",1,CHOICE,_),
 element (CHOICE,LIST,C1),
 chain_terms (mydba,lib,dbdom,library(A1,B1,C1,D1,E1,F1),REF),
 shiftwindow (2),
 clearwindow,
 write ("\n\n  Use the arrow keys to move from line to line to edit text.",
      "  Use back delete\n to eliminate unwanted characters. Press esc to",
      " exit the editor."),
 shiftwindow (1),
 write ("\n Author       : ",B1,
      "\n Title        : ",C1,
      "\n              ",D1,
      "\n Publisher    : ",E1,
      "\n Date         : ",F1),
 cursor (1,17),
 editor (1,17,77,left),!,
 field_str (1,17,60,B2),
 field_str (2,17,60,C2),
 field_str (3,17,60,D2),
 field_str (4,17,60,E2),
 field_str (5,17,60,F2),
 term_replace (mydba,dbdom,REF,library(A1,B2,C2,D2,E2,F2)),
 shiftwindow (2),!,
 act_part_3 (fkey(4),RULENAME).

ref_editor (fkey(4),RULENAME,_):-!,
 shiftwindow (2),
 clearwindow,
 write ("\n\n  Use the arrow keys to move from line to line to edit text.",
      "  Use back delete to \n eliminate unwanted characters. Press esc to",
      " exit the editor."),
 shiftwindow (1),
 clearwindow,
 write ("  RULE : ",RULENAME),
 write ("\n Author       : ",
      "\n Title        : ",
      "\n              ",
      "\n Publisher    : ",
      "\n Date         : "),
 cursor (1,17),
 editor (1,17,77,left),!,
```

```
 field_str (1,17,60,B2),
 field_str (2,17,60,C2),
 field_str (3,17,60,D2),
 field_str (4,17,60,E2),
 field_str (5,17,60,F2),
 write ("\n\n  DO YOU WANT TO KEEP THIS REFERENCE ? "),
 menu (5,25,23,1,["  Yes ","  No  "],"",1,CHOOSE),
 CHOOSE = 1,!,
 lastnumber_ref (1,NO),
 NUM = NO + 1,
 chain_insertz (mydba,lib,dbdom,library(NUM,B2,C2,D2,E2,F2),_),
 shiftwindow (2),
 act_part_3 (fkey(4),RULENAME).

ref_editor (fkey(4),RULENAME,_):-!,
 act_part_3 (fkey(4),RULENAME).

ref_editor (fkey(6),RULENAME,_):-!,
  part_3 (RULENAME).

ref_editor (_,RULENAME,A):-!,
 beep,
 readkey(KEY),!,
 ref_editor (KEY,RULENAME,A).
/*-------------------------------------------------------------------*/
update_3_cond (RULENAME,LIST) :-
 chain_terms(mydba,rul,dbdom,rule(_,_,CSTAR,RULENAME,_,_,_),_),
 not (CSTAR = [[]]),!,
 retractall (snake(_)),
 cut_cond (CSTAR,[],LIST).

update_3_cond (_,LIST):-
 LIST = [].
/*-------------------------------------------------------------------*/
update_3_mtd (RULENAME,LIST) :-
 chain_terms(mydba,rul,dbdom,rule(_,_,_,RULENAME,_,MTDL,_),_),
 not (MTDL = []),!,
 retractall (snake(_)),
 cut_mtd (MTDL,[],LIST).

update_3_mtd (_,LIST):-
 LIST = [].
/*-------------------------------------------------------------------*/
update_3_std (RULENAME,LIST) :-
 chain_terms(mydba,rul,dbdom,rule(_,_,_,RULENAME,_,_,STANDARDL),_),
 not (STANDARDL = []),!,
 retractall (snake(_)),
 cut_std (STANDARDL,[],LIST).

update_3_std (_,LIST):-
 LIST = [].
/*-------------------------------------------------------------------*/
```

```
update_3_ref (RULENAME,LIST) :-
 chain_terms(mydba,rul,dbdom,rule(_,_,_,RULENAME,PUBL,_,_),_),
 not (PUBL = []),!,
 retractall (snake(_)),
 cut_ref (PUBL,[],LIST).

update_3_ref (_,LIST):-
 LIST = [].
/*-----------------------------------------------------------------*/
update_4_cond ([],_) :-
 retractall (snake(_)),
 fail.

update_4_cond ([],_) :-
 chain_terms(mydba,con,dbdom,cond(_,STRING,_),_),
 asserta (snake(STRING)),
 fail.

update_4_cond (LISTIN,LISTFINAL):-
 retract (snake(STRING)),!,
 append ([STRING],LISTIN,LISTOUT),
 update_4_cond (LISTOUT,LISTFINAL).

update_4_cond (LISTIN,LISTFINAL):-
 LISTFINAL = LISTIN.
/*-----------------------------------------------------------------*/
update_4_mtd ([],_) :-
 retractall (snake(_)),
 fail.

update_4_mtd ([],_) :-
 chain_terms(mydba,mtd,dbdom,more_to_do(_,STRING),_),
 asserta (snake(STRING)),
 fail.

update_4_mtd (LISTIN,LISTFINAL):-
 retract (snake(STRING)),!,
 append ([STRING],LISTIN,LISTOUT),
 update_4_mtd (LISTOUT,LISTFINAL).

update_4_mtd (LISTIN,LISTFINAL):-
 LISTFINAL = LISTIN.
/*-----------------------------------------------------------------*/
update_4_std ([],_) :-
 retractall (snake(_)),
 fail.

update_4_std ([],_) :-
 chain_terms(mydba,std,dbdom,standard(_,STRING,_,_,_,_,_),_),
 asserta (snake(STRING)),
 fail.
```

```
update_4_std (LISTIN,LISTFINAL):-
 retract (snake(STRING)),!,
 append ([STRING],LISTIN,LISTOUT),
 update_4_std (LISTOUT,LISTFINAL).

update_4_std (LISTIN,LISTFINAL):-
 LISTFINAL = LISTIN.
/*-----------------------------------------------------------------*/
update_4_ref ([],_) :-
 retractall (snake(_)),
 fail.

update_4_ref ([],_) :-
 chain_terms(mydba,lib,dbdom,library(_,_,TITLE,_,_,_),_),
 asserta (snake(TITLE)),
 fail.

update_4_ref (LISTIN,LISTFINAL):-
 retract (snake(STRING)),!,
 append ([STRING],LISTIN,LISTOUT),
 update_4_ref (LISTOUT,LISTFINAL).

update_4_ref (LISTIN,LISTFINAL):-
 LISTFINAL = LISTIN.
/*-----------------------------------------------------------------*/
 lastnumber_cond (NO,NUMBER):-
  chain_terms (mydba,con,dbdom,cond(NUM,_,_),_),
  NUM > NO,!,
  lastnumber_cond (NUM,NUMBER).

 lastnumber_cond (NO,NUMBER):-!,
  NO = NUMBER.
/*-----------------------------------------------------------------*/
 lastnumber_mtd (NO,NUMBER):-
  chain_terms (mydba,mtd,dbdom,more_to_do(NUM,_),_),
  NUM > NO,!,
  lastnumber_mtd (NUM,NUMBER).

 lastnumber_mtd (NO,NUMBER):-!,
  NO = NUMBER.
/*-----------------------------------------------------------------*/
 lastnumber_std (NO,NUMBER):-
  chain_terms (mydba,std,dbdom,standard(NUM,_,_,_,_,_,_),_),
  NUM > NO,!,
  lastnumber_std (NUM,NUMBER).

 lastnumber_std (NO,NUMBER):-!,
  NO = NUMBER.
/*-----------------------------------------------------------------*/
 lastnumber_rule (NO,NUMBER):-
  chain_terms (mydba,rul,dbdom,rule(NUM,_,_,_,_,_),_),
  NUM > NO,!,
```

```
  lastnumber_rule (NUM,NUMBER).

lastnumber_rule (NO,NUMBER):-!,
  NO = NUMBER.
/*-------------------------------------------------------------------*/
 lastnumber_ref (NO,NUMBER):-
   chain_terms (mydba,lib,dbdom,library(NUM,_,_,_,_,_),_),
   NUM > NO,!,
   lastnumber_ref (NUM,NUMBER).

lastnumber_ref (NO,NUMBER):-!,
  NO = NUMBER.
/*-------------------------------------------------------------------*/
 make_new_cond_list ([HEAD|TAIL],CHOICE,LIST,NEWLIST):-
  not (CHOICE = 1),!,
  appendcstar ([HEAD],LIST,LIST1),
  C = CHOICE - 1,
  make_new_cond_list (TAIL,C,LIST1,NEWLIST).

 make_new_cond_list ([_|TAIL],CHOICE,LIST,NEWLIST):-
  CHOICE = 1,!,
  C = CHOICE - 1,
  make_new_cond_list (TAIL,C,LIST,NEWLIST).

 make_new_cond_list ([],_,LIST,NEWLIST):-
  NEWLIST = LIST.
/*-------------------------------------------------------------------*/
 make_new_mtd_list ([HEAD|TAIL],CHOICE,LIST,NEWLIST):-
  not (CHOICE = HEAD),!,
  append ([HEAD],LIST,LIST1),
  make_new_mtd_list (TAIL,CHOICE,LIST1,NEWLIST).

 make_new_mtd_list ([HEAD|TAIL],CHOICE,LIST,NEWLIST):-
  CHOICE = HEAD,!,
  make_new_mtd_list (TAIL,CHOICE,LIST,NEWLIST).

 make_new_mtd_list ([],_,LIST,NEWLIST):-
  NEWLIST = LIST.
/*-------------------------------------------------------------------*/
make_new_std_list ([HEAD|TAIL],CHOICE,LIST,NEWLIST):-
  not (CHOICE = HEAD),!,
  append ([HEAD],LIST,LIST1),
  make_new_std_list (TAIL,CHOICE,LIST1,NEWLIST).

make_new_std_list ([HEAD|TAIL],CHOICE,LIST,NEWLIST):-
  CHOICE = HEAD,!,
  make_new_std_list (TAIL,CHOICE,LIST,NEWLIST).

make_new_std_list ([],_,LIST,NEWLIST):-
  NEWLIST = LIST.
/*-------------------------------------------------------------------*/
```

```
make_new_ref_list ([HEAD|TAIL],CHOICE,LIST,NEWLIST):-
 not (CHOICE = HEAD),!,
 append ([HEAD],LIST,LIST1),
 make_new_ref_list (TAIL,CHOICE,LIST1,NEWLIST).

make_new_ref_list ([HEAD|TAIL],CHOICE,LIST,NEWLIST):-
 CHOICE = HEAD,!,
 make_new_ref_list (TAIL,CHOICE,LIST,NEWLIST).

make_new_ref_list ([],_,LIST,NEWLIST):-
 NEWLIST = LIST.
/*-----------------------------------------------------------------*/
find_cond_value ([[cn(A),r(B),r(C)]|_],NUMBER,LCFN,LCFM):-
 NUMBER = A,!,
 LCFN = B,
 LCFM = C.

find_cond_value ([_|TAIL],NUMBER,LCFN,LCFM):-!,
 find_cond_value (TAIL,NUMBER,LCFN,LCFM).

find_cond_value ([],_,0,0).
/*-----------------------------------------------------------------*/
 Title_page :-!,
  write ( "\n\n                    Geni Database Editor    "),
  write ("\n\n\n                      version 1.0       "),
  write ("\n\n\n                      written by        "),
  write ("\n\n\n                     J. R. Hoglund      "),
  write ("\n\n\n                        and             "),
  write ("\n\n\n                      R. G. Kirk        "),
  write ("\n\n\n                       VPI & SU         "),
  readkey (_),
clearwindow,
write ("\n      You are about to use the database editor built to be used"),
write ("\n    with the Geni Expert Solver, version 1.0. If this is not the"),
write ("\n    version of Geni that you are using, please consult your user's"),
write ("\n    manual to make sure they are compatible."),
write ("\n      It is worth noting that if the editor is modifing a database"),
write ("\n    and for some reason the program is ended prematurely, like in the"),
write ("\n    case of a power failure, the database would become unusable by "),
write ("\n    the system. So, it is advisable that all databases have backup "),
write ("\n    copies made of them before they are edited."),
write ("\n      It is intended that this program be user friendly. Menues "),
write ("\n    or directions are usually located at the bottom of the screen "),
write ("\n    explaining the types of options advailable to you. If a list "),
write ("\n    of conditions or standards or other things appear at the top "),
write ("\n    of the screen with a light bar, place the light bar over the "),
write ("\n    object that you want to work on and then press the function key"),
write ("\n    for the activity that you are going to execute. The program "),
write ("\n    will automatically pick up the entry that the light bar is on,"),
write ("\n    if it applies to the desired action indicated by the function "),
write ("\n    key that you press."),
write ("\n\n              (Press Any Key To Begin)"),
```

```
readkey (_).
/*-------------------------------------------------------------------*/
 append ([],List,List).
 append ([X|L1],List2,[X|L3]) :-
  append (L1,List2,L3).
/*-------------------------------------------------------------------*/
 appendcstar ([],List,List).
 appendcstar ([[]],List,List).
 appendcstar ([X|L1],List2,[X|L3]) :-
  appendcstar (L1,List2,L3).
/*-------------------------------------------------------------------*/
 append_change ([],_,List):-!,
  List = [].
 append_change ([[cn(A),_,_]|L1],[[cn(A),r(B),r(C)]],[[cn(A),r(B),r(C)]|L3]) :-
  append_change (L1,[[cn(A),r(B),r(C)]],L3).
 append_change ([X|L1],List2,[X|L3]) :-
  append_change (L1,List2,L3).
/*-------------------------------------------------------------------*/
 element (INTEGER,[HEAD|_],ELEMENT):-
  INTEGER = 1,!,
  ELEMENT = HEAD.

 element (INTEGER,[_|TAIL],ELEMENT):-!,
  INTEGER > 1,
  INT = INTEGER - 1,
  element (INT,TAIL,ELEMENT).
/*-------------------------------------------------------------------*/
 elements (INTEGER,[HEAD|_],ELEMENT):-
  INTEGER = 1,!,
  ELEMENT = [HEAD].

 elements (INTEGER,[_|TAIL],ELEMENT):-!,
  INT = INTEGER - 1,
  elements (INT,TAIL,ELEMENT).

elements (_,[],[]).
/*-------------------------------------------------------------------*/
cut_cond ([[cn(CNO),r(_),r(_)]|TAIL],LISTIN,LISTFINAL):-!,
 chain_terms(mydba,con,dbdom,cond(CNO,STRING,_),_),
 asserta (snake(STRING)),
 cut_cond (TAIL,LISTIN,LISTFINAL).

cut_cond ([],LISTIN,LISTFINAL):-
 retract (snake(STRING)),!,
 append ([STRING],LISTIN,LISTOUT),
 cut_cond ([],LISTOUT,LISTFINAL).

cut_cond ([],LISTIN,LISTFINAL):-
 LISTFINAL = LISTIN.

cut_cond ([],[],[]).
/*-------------------------------------------------------------------*/
```

```
cut_mtd ([HEAD|TAIL],LISTIN,LISTFINAL):-!,
 chain_terms(mydba,mtd,dbdom,more_to_do(HEAD,STRING),_),
 asserta (snake(STRING)),
 cut_mtd (TAIL,LISTIN,LISTFINAL).

cut_mtd ([],LISTIN,LISTFINAL):-
 retract (snake(STRING)),!,
 append ([STRING],LISTIN,LISTOUT),
 cut_mtd ([],LISTOUT,LISTFINAL).

cut_mtd ([],LISTIN,LISTFINAL):-
 LISTFINAL = LISTIN.

cut_mtd ([],[],[]).
/*-----------------------------------------------------------------*/
cut_std ([HEAD|TAIL],LISTIN,LISTFINAL):-!,
 chain_terms(mydba,std,dbdom,standard(HEAD,TITLE,_,_,_,_,_),_),
 asserta (snake(TITLE)),
 cut_std (TAIL,LISTIN,LISTFINAL).

cut_std ([],LISTIN,LISTFINAL):-
 retract (snake(STRING)),!,
 append ([STRING],LISTIN,LISTOUT),
 cut_std ([],LISTOUT,LISTFINAL).

cut_std ([],LISTIN,LISTFINAL):-
 LISTFINAL = LISTIN.

cut_std ([],[],[]).
/*-----------------------------------------------------------------*/
cut_ref ([HEAD|TAIL],LISTIN,LISTFINAL):-!,
 chain_terms(mydba,lib,dbdom,library(HEAD,_,TITLE,_,_,_),_),
 asserta (snake(TITLE)),
 cut_ref (TAIL,LISTIN,LISTFINAL).

cut_ref ([],LISTIN,LISTFINAL):-
 retract (snake(STRING)),!,
 append ([STRING],LISTIN,LISTOUT),
 cut_ref ([],LISTOUT,LISTFINAL).

cut_ref ([],LISTIN,LISTFINAL):-!,
 LISTFINAL = LISTIN.

cut_ref ([],[],[]).
/*-----------------------------------------------------------------*/
 test ('y').
 test ('Y').
/*-----------------------------------------------------------------*/
 editor(Row,_,Array,up):-
  Row > 1,!,
  field_attr (Row,17,61,23),
  R = Row - 1,
```

```
 cursor (R,17),
 field_attr (R,17,61,112),
 readkey(KEY),
 editor(R,17,Array,KEY).

editor(Row,_,Array,down):-
 Row<8,!,
 field_attr (Row,17,61,23),
 R = Row + 1,
 cursor (R,17),
 field_attr (R,17,61,112),
 readkey(KEY),
 editor(R,17,Array,KEY).

editor(Row,Column,Array,char(Ch)):-
 Column < Array,!,
 scr_char(Row,Column,Ch),
 C=Column+1,
 cursor (Row,C),
 readkey(KEY),
 editor(Row,C,Array,KEY).

editor(Row,Column,Array,bdel):-
 Column>17,!,
 C=Column-1,
 cursor (Row,C),
 readkey(KEY),
 editor(Row,C,Array,KEY).

editor (Row,Column,Array,right):-
 Column < Array,!,
 C = Column + 1,
 cursor (Row,C),
 readkey (KEY),
 editor (Row,C,Array,KEY).

editor (Row,Column,Array,pgup):-
 Column < Array-5,!,
 C = Column + 5,
 cursor (Row,C),
 readkey (KEY),
 editor (Row,C,Array,KEY).

editor (Row,Column,Array,left):-
 Column < 17,!,
 C = Column - 1,
 cursor (Row,C),
 readkey (KEY),
 editor (Row,C,Array,KEY).

editor(R,_,_,esc):-!,
 field_attr(R,17,61,23).
```

```
 editor(Row,Column,Array,_):-
  sound(5,50),!,
  readkey(KEY),
  editor(Row,Column,Array,KEY).
/*-----------------------------------------------------------------*/
stall :-
 makewindow (9,12,64,"",10,20,3,40),
 write ("     Please Wait !  I'll hurry.").
/*-----------------------------------------------------------------*/
error (NUM) :-
 not (NUM = 0),
 db_close (mydba),
 shiftwindow (1),
 clearwindow,
 window_attr (32),
 write ("\n\n  Error or Corrupt file, executing automatic save "),
 write ("\n\n  Need to load old (F5) or Create a new (F1) database"),
 db_close (mydba),
 run.

error (_) :-
 db_close (mydba),
 exit.
```

Appendix C. Predicates and Code for Expert Program

The predicates will be discussed in a fashion similar to the Turbo Prolog 2.0 Reference Guide (10). This means that they will be listed by name, function performed, what domain types are used in the call to the predicate, which objects must be known and those determined by the predicate in normal operation and under what conditions does the predicate fail. Two standard predicates are also used by the program. One is menu2 that presents a list and returns the position in the list of the item chosen from the list. The other is longmm, an adaptation of longmenu, that allows for list that can be scrolled and returns the position in the list of the item chosen and the key that was pressed to choose the item.

Predicates - Expert Shell

action

Function	Interprets the user's desires as to whether to restart the program or go to the solutions screen in the advent that the database is exhausted. Ends in going back to restart or returning to output.
Declaration	action (RESPONSE)
Domains	(INTEGER)
Flow pattern	(i)

Fails	RESPONSE not one or two

add_answer

Function	Inserts a condition's reference number into the internal database that matches its evaluation, ie true, false, or not known.
Declaration	add_answer (CONDITION_NUMBER, TRUTH_INTEGER)
Domains	(CNO, INTEGER)
Flow patterns	(i,i)
Fails	INTEGER not one, two or three

append

Function	Combines a FIRST_LIST and SECOND_LIST into a COMBINED_LIST.
Declaration	append (FIRST_LIST, SECOND_LIST, COMBINED_LIST)
Domains	(STRINGLIST, STRINGLIST, STRINGLIST)
Flow pattern	(i,i,o)
Fails	Never

bmtdl1

Function	Performs the actual printing for mtdl1.
Declaration	bmtdl1 (LIST_OF_CONDITION_NUMBERS)
Domains	(CSTAR)
Flow pattern	(i)
Fails	Never

bmtdl2

Function	Performs the actual printing for mtdl2.
Declaration	bmtdl2 (LIST_OF_CONDITION_NUMBERS)
Domains	(CSTAR)
Flow pattern	(i)
Fails	Never

bref

Function	Predicate called by ref. Takes the reference list passed to it and prints out the entry in a standard form, with pauses between each entry displayed.
Declaration	bref (REFERENCE_LIST)
Domains	(PUBL)
Flow pattern	(i)

Fails	Never

changeover

Function	Converts the suggested further actions reference number list for a rule and makes it into a list of suggested further actions
Declaration	changeover (REFERENCE_NUMBER_LIST, ACTION_LIST)
Domains	(MTDL, STRINGLIST)
Flow pattern	(i,o)
Fails	Never

conditionprint

Function	Takes a list of condition reference numbers for a rule and prints them out with the user defined values attached to each condition. This is intended to be writing to the printer as there are no pauses between print statements.
Declaration	conditionprint (CONDITION_REF_NO_LIST)
Domains	(CSTAR)
Flow pattern	(i)
Fails	Never

conditionprint1

Function	Takes a list of condition reference numbers for a rule and shows them in a menu that can be scrolled form to the screen, with the user defined values attached to each condition.
Declaration	conditionprint1 (CONDITION_REF_NO_LIST, MENU_LIST)
Domains	(CSTAR, STRINGLIST)
Flow pattern	(i)
Fails	CSTAR = []

copy

Function	Makes the internal database snake the same as the internal database solutions
Declaration	copy
Domains	
Flow pattern	
Fails	Never

copyit

Function	Makes a the internal database solutions the same the internal database ordered
Declaration	copyit
Domains	
Flow pattern	
Fails	Never

listlength

Function	Determines the number of conditions LENGTH that are assigned to a rule.
Declaration	listlength (CONDITIONLIST, LENGTH)
Domains	(CSTAR, INTEGER)
Flow patterns	(i,o)
Fails	Never

listprint

Function	List out the suggested further actions for a rule assuming it is the reason of the difficulty. It is intended to be used with the printer as there are no pauses between output lines.
Declaration	listprint (SUGGESTED_FURTHER_ACTION_REF_NO_LIST)
Domains	(MTDL)
Flow pattern	(i)
Fails	Never

makelist

Function	Determines the RULE_NUMBER of the rule located in the NUMBERth location in the solutions list of the solution menu.
Declaration	makelist (NUMBER, RULE_NUMBER)
Domains	(INTEGER, RNO)
Flow pattern	(i,o)
Fails	less than NUMBERth elements in snake

makeoutput

Function	Creates the list LIST of solutions and their corresponding confidence factors to be used in the scrolling menu on the solutions screen.
Declaration	makeoutput ([], LIST)
Domains	(STRINGLIST, STRINGLIST)

Flow pattern	(i,o)
Fails	Never

mtdl1

Function	Writes out what further actions should be taken to more solidly prove or disprove a hypothesis about why the machine is misbehaving. This intended to be used for screen output as pauses have been included between each suggested further action.
Declaration	mtdl1 (RULE_NUMBER_NEEDING_FURTHER_TESTING)
Domains	(RNO)
Flow pattern	(i)
Fails	Never

mtdl2

Function	Writes out what further actions should be taken to more solidly prove or disprove a hypothesis about why the machine is misbehaving. This intended to be used for printer output as suggestions are printed without pauses.
Declaration	mtdl2 (RULE_NUMBER_NEEDING_FURTHER_TESTING)
Domains	(RNO)
Flow pattern	(i)
Fails	Never

order

Function	Reorders the solutions so that they are ordered from highest to lowest in terms of final confidence factors.
Declaration	order (RULE_NUMBER, CONFIDENCE_FACTOR)
Domains	(RNO, REAL)
Flow pattern	(i,i)
Fails	Never

output

Function	Prints out the menu for the solutions screen. Ends in reoutput.
Declaration	output
Domains	
Flow pattern	
Fails	No solutions have been found

pref

Function	Prints out the reference list. It is intended to be used as formatted output to the printer, as the printing is not paused between consecutive entries.
Declaration	pref (REFERENCE_LIST)
Domains	(PUBL)
Flow pattern	(i)
Fails	Never

printstandard

Function	Prints out the standards that are related to a rule. This is intended to be used as a formatted output to a printer as there are no pauses between standards being printed.
Declaration	printstandard (STANDARD_REF_NO_LIST)
Domains	(STANDARDL)
Flow pattern	(i)
Fails	Never

prompt

Function	Determines the value of a condition by seeing if has been answered before, and if not it prompts the user for the conditions value (or truth). It also determines if the rule under evaluation contains enough truth to still consider as a possible solution.
Declaration	prompt (RULE_NUMBER, CONDITION_LIST, CURRENT_CONFIDENCE, MINIMUM_CONFIDENCE_ALLOWABLE, CONDITION_LIST_LENGTH, CURRENT_SUM_OF_NO'S, CURRENT_SUM_OF_MAYBE'S, NUMBER_OF_NO'S)
Domains	(RNO, CSTAR, REAL, REAL, INTEGER, REAL, REAL, INTEGER)
Flow patterns	(i,i,o,i,i,o,o,o)
Fails	Never

ref

Function	Prints out the references for a rule
Declaration	ref (RULE_NUMBER)
Domains	(RNO)
Flow pattern	(i)
Fails	Never

reoutput

Function	Operates the scrolling menu of the solutions screen and takes the user response and ends by passing these to resp
Declaration	reoutput (STARTING_LINE_NUMBER, MENU_LIST)
Domains	(INTEGER, STRINGLIST)
Flow pattern	(i,i)
Fails	MENU_LIST = []

resp

Function	Processes the user request for information on a particular rule on the solutions screen. Ends out in reoutput, exit, run, or by failing back to search to find other solutions.
Declaration	resp (fkey(INTEGER), RULE_NUMBER, SOLUTIONS_LIST, CURRENT_LINE_ON_SOLUTIONS_SCREEN)
Domains	(KEY, RNO, STRINGLIST, INTEGER)
Flow pattern	(i,i,i,i)
Fails	RULE_NUMBER not in database

restart

Function	Clears the internal databases that are changed during the expert shell's operation and determines the minimum confidence that is being considered as meaningful by the program. Ends in output and also calls search.
Declaration	restart
Domains	
Flow patterns	
Fails	Never

run

Function	Chooses the database that will be used. Ends in restart.
Declaration	run
Domains	
Flow patterns	
Fails	Never

search

Function	Evaluates the conditions of rules until a rule is found with a final confidence greater than the allowable minimum MINIMUM as determined in restart. Ends in restart if no solutions are found, or a restart is wanted after the database is exhausted.
Declaration	search (MINIMUM)
Domains	(REAL)
Flow patterns	(i)
Fails	If the user wants to continue when the program finds an acceptable solution. In this case it repeats to itself, or if the database is exhausted, gives the option to return to the solutions screen of start a new search

standardprint

Function	Prints out the standards that are related to a rule one at a time.
Declaration	standardprint (STANDARD_REF_NO_LIST)
Domains	(STANDARDL)
Flow pattern	(i)
Fails	Never

test

Function	Determines if the key pressed was Y or y.
Declaration	test (RESPONSE)
Domains	(CHAR)
Flow pattern	(i)
Fails	Key pressed not Y or y.

title_page

Function	Prints out introductory information to the current window
Declaration	title_page
Domains	
Flow patterns	
Fails	Never

Code - Expert Shell

```
code = 2500

 include "menu2.pro"
 include "longmm.pro"

DOMAINS

 db_selector = mydba
 dbdom = rule (RNO,REAL,CSTAR,STRING,PUBL,MTDL,STANDARDL);
      cond (CNO,STRING,STRING);
      library (INTEGER,STRING,STRING,STRING,STRING,STRING);
      more_to_do (MTDNO,STRING);
      standard (INTEGER,STRING,STRING,STRING,STRING,STRING,STRING)

 MTD,AUTHOR,TITLE,PUBLISH,PDATE,FIXES,SUMMARY,SOURCE = STRING
 RNO,CNO,PUB,MTDNO,STANDARD = INTEGER

 CSTAR = CONDITIONL*
 CONDITIONL = C*
 C = cn(CNO);r(REAL)
 RESPONCEL = INTEGER*
 PUBL = INTEGER*
 MTDL = MTDNO*
 RNOL = RNO*
 STANDARDL = INTEGER*
 file = file1; file2

DATABASE
 true_conditions (CNO)
 false_conditions (CNO)
 maybe (CNO)
 solutions (RNO,REAL)
 ordered (RNO,REAL)
 snake (RNO,REAL)

PREDICATES

title_page
run
 restart
  search (REAL)
   listlength (CSTAR,INTEGER)
   prompt (RNO,CSTAR,REAL,REAL,INTEGER,REAL,REAL,INTEGER)
    add_answer (CNO,INTEGER)
   action (INTEGER)
  output
   order (RNO,REAL)
   copyit
```

```
    makeoutput (STRINGLIST,STRINGLIST)
     append (STRINGLIST,STRINGLIST,STRINGLIST)
    reoutput (INTEGER,STRINGLIST)
     copy
     makelist (INTEGER,RNO)
     resp (KEY,RNO,STRINGLIST,INTEGER)
      ref (RNO)
       bref (PUBL)
      pref (PUBL)
      changeover (MTDL,STRINGLIST)
      listprint (MTDL)
      conditionprint1 (CSTAR,STRINGLIST)
      conditionprint (CSTAR)
      standardprint (STANDARDL)
      printstandard (STANDARDL)
      mtd1 (RNO)
       bmtd1 (CSTAR)
      mtd2 (RNO)
       bmtd2 (CSTAR)

test (CHAR)

GOAL
 trace(off),
 makewindow (1,31,64,"",0,0,25,80),
 title_page,
 run.

CLAUSES
 run:-
  clearwindow,
  write ("\n          Choose the data base with which you want to work"),
  write ("\n          Press F4 if you need to change the drive source"),
  makewindow (3,7,64,"",5,10,8,60),
  dir ("","*.dba",DosFileName,1,1,1),
  db_open (mydba,DosFileName,in_file),
  removewindow,
  clearwindow,!,
  restart.

restart:-
 repeat,
  retractall (true_conditions(_)),
  retractall (false_conditions(_)),
  retractall (solutions(_,_)),
  retractall (maybe(_)),
  retractall (snake(_,_)),
  retractall (ordered(_,_)),
  clearwindow,
  write ("\n"),
  write ("\n\n\n  What is the minimum confidence factor that you wish to consider ?"),
  write ("\n\n       (I suggest 0 < VALUE < 1, for example = = > 0.1) "),
```

```
  write ("\n\n    Please type your value here = = > "),
  readreal (MINIMUM),
  search (MINIMUM),
  output.
/*-----------------------------------------------------------------*/
search (MINIMUM) :-
  chain_terms (mydba,rul,dbdom,rule(RNO,CONFIDENCE,CONS,_,_,_,_),_),
  listlength(CONS,LISTLENGTH),
  clearwindow,
  prompt (RNO,CONS,CONFIDENCE,MINIMUM,LISTLENGTH,0,0,0),
  window_attr(15),
  write ("\n\n  A solution has been determined .....\n ..\n .."),
  write ("\n ..\n ..\n ..\n ..\n ..\n ..\n ..\n .."),
  write ("\n ..\n .. SHOULD I CONTINUE THE SEARCH ?\n"),
  ROW = 14,
  COLUMN = 35,
  menu (ROW,COLUMN,23,1,["  Yes "," No  "],"",1,CHOICE),
  window_attr(31),
  clearwindow,
  CHOICE = 2.

search (_):-!,
repeat,
  sound(10,500),
  clearwindow,
  write ("\n\n  YOU HAVE REACHED THE END OF THE DATABASE."),
  write ("\n\n  DO YOU WANT TO RETURN TO THE SOLUTIONS SCREEN OR"),
  write ("\n\n  REINITALIZE THE SEARCH AND START AGAIN ?"),
  menu (10,24,4,1,[" RETURN TO SOLUTIONS SCREEN ",
            "    START NEW SEARCH     "],"WARNING",1,CHOICE),
  action (CHOICE).

 action (1):-!,clearwindow.
 action( 2):-!,
   restart.
/*-----------------------------------------------------------------*/
 listlength ([],0).

 listlength ([_|TAIL],LENGTH) :-
  listlength(TAIL,TAILLENGTH),
  LENGTH = TAILLENGTH + 1.
/*-----------------------------------------------------------------*/
 prompt (RNO,[[cn(HEAD),_,_]|TAIL],CONFIDENCE,MINIMUM,
     LISTLENGTH,SUMN,SUMM,NN) :-
  true_conditions (HEAD),!,
  prompt (RNO,TAIL,CONFIDENCE,MINIMUM,LISTLENGTH,SUMN,SUMM,NN).

 prompt (RNO,[[cn(HEAD),_,r(MAYBE)]|TAIL],CO,MINIMUM,LL,SN,SM,NN) :-
  maybe (HEAD),
  S = SM + MAYBE,
  P = (CO*(S)/(LL+1)),
  QR = (CO*NN/(LL-0.999)),
```

```
 QR > = 0,
 Q = SQRT (QR),
 MINIMUM < = CO - P*P - Q*SN/(NN + 0.001),!,
 prompt (RNO,TAIL,CO,MINIMUM,LL,SN,S,NN).

prompt (RNO,[[cn(HEAD),r(NO),_]¦TAIL],CO,MINIMUM,LL,SN,SM,NN) :-
 false_conditions (HEAD),
 N = NN + 1,
 S = SN + NO,
 QR = (CO*NN/(LL-0.999)),
 QR > = 0,
 Q = SQRT (QR),
 P = (CO*(SM)/(LL + 1)),
 MINIMUM < = CO - P*P - Q*S/(N + 0.001),!,
 prompt (RNO,TAIL,CO,MINIMUM,LL,S,SM,N).

prompt (RNO,[[cn(HEAD),r(V),r(E)]¦TAIL],CO,MINIMUM,LL,SN,SM,NN) :-
 not(true_conditions(HEAD)),
 not(false_conditions (HEAD)),
 not(maybe(HEAD)),!,
 chain_terms (mydba,con,dbdom,cond (HEAD,STRING,_),_),
 writef ("\n Is it true that .....\n\n % ?",STRING),
 ROW    = 14,
 COLUMN = 34,
 menu (ROW,COLUMN,4,12,["   Yes","   No","Don't Know"],"",1,CHOICE),
 clearwindow,
 add_answer (HEAD,CHOICE),!,
 prompt (RNO,[[cn(HEAD),r(V),r(E)]¦TAIL],CO,MINIMUM,LL,SN,SM,NN).

prompt (RNO,[],CO,MN,LL,SN,SM,NN) :-
 QR = (CO*NN/(LL-0.999)),
 QR > = 0,
 Q = SQRT (QR),
 P = (CO*(SM)/(LL + 1)),
 VITA = CO - P*P - Q*SN/(NN + 0.001),
 VITA > = MN,!,
 assertz(solutions(RNO,VITA)).
/*-----------------------------------------------------------------*/
add_answer (HEAD,1) :-!,
 asserta (true_conditions(HEAD)).

add_answer (HEAD,2) :-!,
 asserta (false_conditions(HEAD)).

add_answer (HEAD,3) :-!,
 asserta (maybe(HEAD)).
/*-----------------------------------------------------------------*/
output :-
 clearwindow,
 solutions (RN,CON),
 order (RN,CON),
 copyit,
```

```
 makeoutput ([],LIST),
 makewindow (1,31,64,"",0,0,25,80),
 makewindow (2,31,64,"",9,0,16,80),
 write ("    I have concluded the above ; for more information you may wish to move"),
 write ("\n the lighted bar over a machine difficulty and press the ACTION KEY indicated"),
 write ("\n\n KEY       ACTION                KEY       ACTION "),
 write ("\n ___  ____________________________   ___ ____________________________"),
 write ("\n"),
 write ("\n  F1  suggest REFERENCES            F2  PRINT references"),
 write ("\n  F3  recommend FIXES               F4  PRINT recommended fixes"),
 write ("\n  F5  give condition list           F6  PRINT condition list"),
 write ("\n  F7  suggest STANDARDS             F8  PRINT standards"),
 write ("\n  F9  additional information required   F10 PRINT additional information"),
 write ("\n  bdel to print the summary list above   ESC to exit    "),
 write ("\n          (press Ctrl-home to CONTINUE search)"),
 write ("\n   (press Ctrl-PgUp to choose a new database and begin again)"),
 makewindow (9,31,64,"",0,0,3,80),
 write ("\n   I have concluded that your machine difficulty may be due to :"),
 makewindow (3,7,64,"",2,0,8,80),!,
 reoutput (0,LIST).
/*-----------------------------------------------------------------*/
reoutput (LINE,LIST) :-!,
 copy,!,
 ROW = 3,
 COLUMN = 1,
 LENGTH = 6,
 longmm(ROW,COLUMN,LENGTH,7,_,LIST,"",LINE,CHOICE,KEY),!,
 makelist(CHOICE,Ruleno),!,
 resp (KEY,Ruleno,LIST,CHOICE).
/*-----------------------------------------------------------------*/
order (_,CON) :-
 solutions (RN1,CON1),
 CON1 > CON,
 order (RN1,CON1).

order (RN,CON) :-
 assertz (ordered (RN,CON)),
 retract (solutions (RN,CON)),
 solutions (RN1,CON1),
 order (RN1,CON1).

order (_,_) :-!.
/*-----------------------------------------------------------------*/
copyit :-
  retractall (solutions (_,_)),
  fail.

copyit :-
  ordered (A,B),
  asserta (solutions (A,B)),
  fail.
```

```
 copyit:-!.
/*-----------------------------------------------------------------*/
 makeoutput (LIST,FINAL_LIST) :-
  ordered(ELEMENT,CONFIDENCE),!,
  chain_terms(mydba,rul,dbdom,rule (ELEMENT,_,_,REASON,_,_,_),_),
  retract (ordered (ELEMENT,_)),
  format (STRING,"  %-50  %20.2",REASON,CONFIDENCE),
  ADDITIVE = [STRING],
  append (LIST,ADDITIVE,NEWLIST),
  makeoutput (NEWLIST,FINAL_LIST).

 makeoutput (LIST,FINAL_LIST):-!,
  LIST = FINAL_LIST.
/*-----------------------------------------------------------------*/
 copy :-
  retractall (snake(_,_)),
  fail.

 copy :-
  solutions (A,B),
  asserta (snake(A,B)),
  fail.

 copy :-!.
/*-----------------------------------------------------------------*/
 makelist (CHOICEline,A):-
  not (CHOICEline < = 1),!,
  retract (snake(_,_)),
  N = CHOICEline - 1,
  makelist (N,A).

 makelist (_,RESP) :-!,
  snake (RESP,_).
/*-----------------------------------------------------------------*/
 resp (fkey(1),RESP,LIST,LINE) :-!,
  ref (RESP),!,
  reoutput(LINE,LIST).

 resp (fkey(2),RESP,LIST,LINE) :-!,
  writedevice (printer),
  chain_terms (mydba,rul,dbdom,rule (RESP,_,_,REASON,PUBS,_,_),_),
  writef ("\nSuggested references for % :\n",REASON),
  pref (PUBS),
  write ("\n\n                    "),
  writedevice (screen),!,
  reoutput(LINE,LIST).

 resp (fkey(3),RESP,LIST,LINE) :-!,
  makewindow (5,31,64,"",0,0,24,80),
  chain_terms (mydba,rul,dbdom,rule (RESP,_,_,REASON,_,FIXES,_),_),
  writef ("\n    Possible action items for  % :\n",REASON),
  changeover (FIXES,[]),
```

```
 removewindow (5,1),!,
 reoutput(LINE,LIST).

resp (fkey(4),RESP,LIST,LINE) :-!,
 writedevice (printer),
 chain_terms (mydba,rul,dbdom,rule (RESP,_,_,REASON,_,FIXES,_),_),
 writef ("\n\nPossible action items for  % :\n",REASON),
 listprint(FIXES),
 write ("\n\n                          "),
 writedevice (screen),!,
 reoutput(LINE,LIST).

resp (fkey(5),RESP,LIST,LINE) :-!,
 chain_terms (mydba,rul,dbdom,rule (RESP,_,CONDL,REASON,_,_,_),_),
 makewindow (5,31,64,"Responses were Y = Yes, D = Don't Know, N = No",0,0,24,80),
 writef ("\n The conditions used to prove % are :\n",REASON),
 conditionprint1 (CONDL,[]),
 removewindow (5,1),!,
 reoutput(LINE,LIST).

resp (fkey(6),RESP,LIST,LINE) :-!,
 writedevice (printer),
 chain_terms (mydba,rul,dbdom,rule (RESP,_,CONDL,REASON,_,_,_),_),
 writef ("\n\n The conditions used to prove % are :\n",REASON),
 conditionprint (CONDL),
 write ("\n\n                  "),
 writedevice (screen),!,
 reoutput(LINE,LIST).

resp (fkey(7),RESP,LIST,LINE) :-!,
 chain_terms (mydba,rul,dbdom,rule (RESP,_,_,REASON,_,_,STANDARDL),_),
 makewindow (6,31,64,"",0,0,24,80),
 writef ("\n        The standard(s) relating to % is/are :",REASON),
 cursor (21,10),
 write ("       Press any key to continue"),
 makewindow (5,31,64,"",3,0,19,80),
 standardprint (STANDARDL),
 removewindow (5,1),
 removewindow (6,1),!,
 reoutput(LINE,LIST).

resp (fkey(8),RESP,LIST,LINE) :-!,
 chain_terms (mydba,rul,dbdom,rule (RESP,_,_,REASON,_,_,STANDARDL),_),
 not(STANDARDL = []),!,
 writedevice (printer),
 writef ("\n        The standard(s) relating to % is/are :",REASON),
 printstandard (STANDARDL),
 write ("\n\n                          "),
 writedevice (screen),!,
 reoutput(LINE,LIST).

resp (fkey(9),RESP,LIST,LINE) :-!,
```

```
 makewindow (6,31,64,"",0,0,24,80),
 mtd1 (RESP),
 clearwindow,
 write ("\n\n    Press any key to continue"),
 readkey(_),!,
 removewindow (6,1),!,
 reoutput(LINE,LIST).

resp (fkey(10),RESP,LIST,LINE) :-!,
 writedevice (printer),
 mtd2 (RESP),
 writedevice (screen),!,
 reoutput(LINE,LIST).

resp (esc,_,_,_):-
 makewindow (1,15,70,"y or n",8,33,3,17),
 write ("Are You Sure ?"),
 readchar (A),
 removewindow,
 test (A),!,
 db_close (mydba),
 removewindow,
 removewindow,
 removewindow,
 removewindow,
 removewindow,
 text,
 exit.

resp (esc,_,LIST,LINE) :-!,
 reoutput (LINE,LIST).

resp (ctrlhome,_,_,_):-!,
 removewindow,
 removewindow,
 removewindow,
 fail.

resp (bdel,_,_,_):-
 writedevice (printer),
 write ("\n \n"),
 copy,
 snake (RNO,CONFIDENCE),
 chain_terms (mydba,rul,dbdom,rule (RNO,_,_,REASON,_,_,_),_),
 writef ("  %-50  %20.2\n",REASON,CONFIDENCE),
 fail.

resp (bdel,_,LIST,LINE):-!,
 write ("\n \n "),
 writedevice (screen),
 reoutput(LINE,LIST).
```

```
resp (ctrlpgup,_,_,_):-!,
 removewindow,
 removewindow,
 removewindow,
 db_close (mydba),
 run.

resp (ctrlpgup,_,_,_):-
 exit.

resp (_,RESP,A,B):-!,
 beep,
 readkey(KEY),
 resp (KEY,RESP,A,B).
/*-----------------------------------------------------------------*/
ref(RESP):-!,
 makewindow (1,31,64,"",0,0,24,80),
 chain_terms (mydba,rul,dbdom,rule (RESP,_,_,REASON,PUBS,_,_),_),
 writef ("        References for % :",REASON),
 cursor (21,10),
 write ("press any key for next reference"),
 makewindow (5,31,64,"",2,0,20,80),
 write ("\n\n"),
 bref (PUBS),
 removewindow (5,1),
 removewindow (1,1).

ref(_):-!.
/*-----------------------------------------------------------------*/
bref ([PUB|REST]):-!,
 chain_terms (mydba,lib,dbdom,library (PUB,AUTHOR,TITLE1,TITLE2,PUBLISH,PDATE),_),
 writef ("\nPaper  : %\n       %\nAuthors : %\nPub. by : %\nDate   : %\n",
        TITLE1,TITLE2,AUTHOR,PUBLISH,PDATE),
 readkey(_),
 bref (REST).

bref      ([]):-!.
/*-----------------------------------------------------------------*/
pref ([PUB|REST]):-!,
 chain_terms (mydba,lib,dbdom,library (PUB,AUTHOR,TITLE1,TITLE2,PUBLISH,PDATE),_),
 writef ("\nPaper  : %\n       %\nAuthors : %\nPub. by : %\nDate   : %\n",
        TITLE1,TITLE2,AUTHOR,PUBLISH,PDATE),
 pref (REST).

pref      ([]):-!.
/*-----------------------------------------------------------------*/
listprint ([HEAD|TAIL]) :-!,
  chain_terms (mydba,mtd,dbdom,more_to_do(HEAD,FIELD),_),
  writef ("\n  * %",FIELD),
  listprint (TAIL).

listprint ([]):-!.
```

```
/*-----------------------------------------------------------------*/
conditionprint1 ([[cn(HEAD),_,_]|TAIL],LIST) :-
 true_conditions(HEAD),
 chain_terms (mydba,con,dbdom,cond(HEAD,FIELD,_),_),
 str_len (FIELD,CONSTANT),
 CONSTANT < 78,!,
 concat (" Y * ",FIELD,ENTRY),
 ADDITIVE = [ENTRY],
 append (LIST,ADDITIVE,NEWLIST),
 conditionprint1 (TAIL,NEWLIST).

conditionprint1 ([[cn(HEAD),_,_]|TAIL],LIST) :-
 true_conditions(HEAD),!,
 chain_terms (mydba,con,dbdom,cond(HEAD,FIELD,_),_),
 concat (" Y * ",FIELD,ENTRY),
 frontstr (78,ENTRY,PART1,PART2),
 concat ("    ",PART2,PART3),
 ADDITIVE = [PART1,PART3],
 append (LIST,ADDITIVE,NEWLIST),
 conditionprint1 (TAIL,NEWLIST).

conditionprint1 ([[cn(HEAD),_,_]|TAIL],LIST) :-
 false_conditions(HEAD),
 chain_terms (mydba,con,dbdom,cond(HEAD,FIELD,_),_),
 str_len (FIELD,CONSTANT),
 CONSTANT < 78,!,
 concat (" N * ",FIELD,ENTRY),
 ADDITIVE = [ENTRY],
 append (LIST,ADDITIVE,NEWLIST),
 conditionprint1 (TAIL,NEWLIST).

conditionprint1 ([[cn(HEAD),_,_]|TAIL],LIST) :-
 false_conditions(HEAD),!,
 chain_terms (mydba,con,dbdom,cond(HEAD,FIELD,_),_),
 concat (" N * ",FIELD,ENTRY),
 frontstr (78,ENTRY,PART1,PART2),
 concat ("    ",PART2,PART3),
 ADDITIVE = [PART1,PART3],
 append (LIST,ADDITIVE,NEWLIST),
 conditionprint1 (TAIL,NEWLIST).

conditionprint1 ([[cn(HEAD),_,_]|TAIL],LIST) :-
 maybe(HEAD),
 chain_terms (mydba,con,dbdom,cond(HEAD,FIELD,_),_),
 str_len (FIELD,CONSTANT),
 CONSTANT < 78,!,
 concat (" D * ",FIELD,ENTRY),
 ADDITIVE = [ENTRY],
 append (LIST,ADDITIVE,NEWLIST),
 conditionprint1 (TAIL,NEWLIST).

conditionprint1 ([[cn(HEAD),_,_]|TAIL],LIST) :-
```

```
 maybe(HEAD),!,
 chain_terms (mydba,con,dbdom,cond(HEAD,FIELD,_),_),
 concat (" D * ",FIELD,ENTRY),
 frontstr (78,ENTRY,PART1,PART2),
 concat ("    ",PART2,PART3),
 ADDITIVE = [PART1,PART3],
 append (LIST,ADDITIVE,NEWLIST),
 conditionprint1 (TAIL,NEWLIST).

conditionprint1 ([[cn(HEAD),_,_]|TAIL],LIST) :-
 maybe(HEAD),!,
 chain_terms (mydba,con,dbdom,cond(HEAD,FIELD,_),_),
 concat (" D * ",FIELD,ENTRY),
 ADDITIVE = [ENTRY],
 append (LIST,ADDITIVE,NEWLIST),
 conditionprint1 (TAIL,NEWLIST).

conditionprint1 ([],LIST):-!,
 Row = 4,
 COLUMN = 0,
 longmm(ROW,COLUMN,10,7,64,LIST,"press any Fkey or Esc to continue",1,_,_).
/*-------------------------------------------------------------------*/
conditionprint ([[cn(HEAD),_,_]|TAIL]) :-
 true_conditions(HEAD),!,
 chain_terms (mydba,con,dbdom,cond(HEAD,FIELD,_),_),
 writef ("\n Y * %",FIELD),
 conditionprint (TAIL).

conditionprint ([[cn(HEAD),_,_]|TAIL]) :-
 maybe(HEAD),!,
 chain_terms (mydba,con,dbdom,cond(HEAD,FIELD,_),_),
 writef ("\n D * %",FIELD),
 conditionprint (TAIL).

conditionprint ([[cn(HEAD),_,_]|TAIL]) :-
 false_conditions(HEAD),!,
 chain_terms (mydba,con,dbdom,cond(HEAD,FIELD,_),_),
 writef ("\n N * %",FIELD),
 conditionprint (TAIL).

conditionprint ([]):-!.
/*-------------------------------------------------------------------*/
standardprint ([HEAD|TAIL]) :-!,
 chain_terms (mydba,std,dbdom,standard(HEAD,TITLE,SOURCE,PDATE,
       SUMMARY1,SUMMARY2,SUMMARY3),_),
 clearwindow,
 write ("\n\n"),
 writef ("\nTitle or Number : %",TITLE),
 writef ("\nSource       : %",SOURCE),
 writef ("\nDate         : %",PDATE),
 writef ("\nSummary of Standard :\n\n   %\n   %\n   %\n",
     SUMMARY1,SUMMARY2,SUMMARY3),
```

```
 readkey(_),
 standardprint (TAIL).

standardprint ([]):-!.
/*-------------------------------------------------------------------*/
printstandard ([HEAD|TAIL]) :-!,
 chain_terms (mydba,std,dbdom,standard(HEAD,TITLE,SOURCE,PDATE,
        SUMMARY1,SUMMARY2,SUMMARY3),_),
 write ("\n\n"),
 writef ("\nTitle or Number : %",TITLE),
 writef ("\nSource       : %",SOURCE),
 writef ("\nDate         : %",PDATE),
 writef ("\nSummary of Standard :\n\n    %\n    %\n    %\n",
     SUMMARY1,SUMMARY2,SUMMARY3),
 printstandard (TAIL).

printstandard ([]):-!.
/*-------------------------------------------------------------------*/
mtd1 (RESP):-!,
 write ("\n\n"),
 chain_terms (mydba,rul,dbdom,rule (RESP,_,CONDITIONS,REASON,_,_,_),_),
 makewindow (7,31,64,"",0,0,24,80),
 writef ("Suggested action(s) to prove hypothysis of % :\n",REASON),
 cursor (21,8),
 write ("*****************************************************************"),
 makewindow (8,31,64,"",3,0,19,80),
 bmtd1 (CONDITIONS),
 removewindow (8,1),
 removewindow (7,1).
/*-------------------------------------------------------------------*/
 bmtd1 ([[cn(MTDNO),_,_]|REST]):-
  maybe (MTDNO),!,
  clearwindow,
  chain_terms (mydba,con,dbdom,cond (MTDNO,COND,MTD),_),
  writef ("\nYou didn't know %,\n    So %\n",COND,MTD),
  write ("\n\n                    (press any key for next item)"),
  readchar(_),
  bmtd1 (REST).

 bmtd1 ([_|REST]):-
  bmtd1 (REST).

 bmtd1 ([]):-!.
/*-------------------------------------------------------------------*/
mtd2 (RESP):-!,
 chain_terms (mydba,rul,dbdom,rule (RESP,_,CONDITIONS,REASON,_,_,_),_),
 writef ("Suggested action(s) to prove hypothysis of % :\n",REASON),
 bmtd2 (CONDITIONS),
 write ("\n").
/*-------------------------------------------------------------------*/
 bmtd2 ([[cn(MTDNO),_,_]|REST]):-
  maybe (MTDNO),!,
```

```
  clearwindow,
  chain_terms (mydba,con,dbdom,cond (MTDNO,COND,MTD),_),
  writef ("\nYou didn't know %,\n    So %\n",COND,MTD),
  bmtd2 (REST).

 bmtd2 ([_|REST]):-
  bmtd2 (REST).

 bmtd2 ([]):-!.
/*---------------------------------------------------------------------*/
 test ('y').
 test ('Y').
/*---------------------------------------------------------------------*/
 title_page :-!,
  write ("\n\n            An Expert System for Turbomachinery "),
  write ("\n\n\n              written in Turbo Prolog     "),
  write ("\n\n\n                  by            "),
  write ("\n\n\n                James R. Hoglund        "),
  write ("\n\n                R. Gordon Kirk          "),
  write ("\n\n\n                VPI & SU          "),
  write ("\n\n                Rotor Dynamics Lab       "),
  write ("\n\n\n                Version 1.0          "),
  readchar(_),
  clearwindow,
write ("\n       You have come to the best available source of knowledge  "),
write ("\n    for your current problem. In order for me to help you, I must "),
write ("\n    ask you some questions which you will answer by moving the    "),
write ("\n    cursor box over your answer and pressing the ENTER key. I can "),
write ("\n    offer one or more likely solutions to your problem and also give"),
write ("\n    you an idea of other likely possibilities which will require that"),
write ("\n    you obtain additional information that I will suggest to you. "),
write ("\n\n     Any time you wish to review my solutions please answer NO"),
write ("\n    to my question -DO YOU WISH TO CONTINUE- = = = > > NO          "),
write ("\n\n     If a possible solution I report seems likely to be the  "),
write ("\n    cause, I will give you my recommendations/changes to help solve"),
write ("\n    your problem. Simply press the F3 key for the recommendation, "),
write ("\n    or F4 for a copy on the printer."),
write ("\n\n     If you want to read more literature or know the source of"),
write ("\n    my information or the occurrence of similar problems, simply  "),
write ("\n    press the F1 key when the cursor box is on the answer of interest,"),
write ("\n    or F2 for a copy of the references."),
write ("\n                                    "),
write ("\n                        (press any key to continue)"),
  readchar(_),
  clearwindow,
write ("\n       To see the condition list with indicators of your response"),
write ("\n    press F5. Press F6 to print this to the printer. "),
write ("\n\n     Press F9 to see additional information required to better"),
write ("\n    prove a hypothysis. Press F10 to print the information."),
write ("\n\n     Press F7 to see related standards to the vibration condition."),
write ("\n    Press F8 to see the related standards printed."),
write ("\n\n     To continue the search for more solutions simply press the"),
```

```
write ("\n    CTRL-Home key combination.                              "),
write ("\n\n      If you have had enough discussion about your problem    "),
write ("\n    and desire no further help from me, press the ESC key when the  "),
write ("\n    solution window is active.                              "),
write ("\n                                                            "),
write ("\n          Give me the facts, I will do the thinking.        "),
write ("\n                                                            "),
write ("\n                                (press any key to continue) "),
  readchar(_).
/*-----------------------------------------------------------------*/
 changeover ([],[]).

 changeover ([HEAD|TAIL],STRAIGHT):-
  chain_terms (mydba,mtd,dbdom,more_to_do (HEAD,C),_),!,
  ADDITIVE = [C],
  append (STRAIGHT,ADDITIVE,NEWSTRAIGHT),
  changeover (TAIL,NEWSTRAIGHT).

 changeover ([],STRAIGHT):-!,
  ROW = 4,
  COLUMN = 0,
  longmm(ROW,COLUMN,10,7,64,STRAIGHT,"press any Fkey or Esc to continue",1,_,_).
/*-----------------------------------------------------------------*/
 append ([],List,List).
 append ([X|L1],List2,[X|L3]) :-
   append (L1,List2,L3).
/*-----------------------------------------------------------------*/
```

The vita has been removed from the scanned document