1991

A diagnostic expert system for recipe book generating system

Shahida Mahmood Parvez
Lehigh University

AUTHOR:
Parvez, Shahida M.

TITLE:A Diagnostic Expert System For Recipe Book Generating System

DATE: January 1992

A DIAGNOSTIC EXPERT SYSTEM FOR RECIPE BOOK GENERATING SYSTEM

by

Shahida Mahmood Parvez

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Computer Science

Lehigh University

1991

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

11/15/91

(date)

Professor in Charge

Chairman of Department

CONTENTS

ABSTRACT

In a manufacturing plant, recipe books accompany the product through the manufacturing process. These books contain step by step instructions on how to manufacture the product and are used by manufacturing shop operators to perform operations on the product. The recipe books provided to the shop operators have to be up-to-date and accurate to manufacture a quality and reliable product. Otherwise, scrap will be generated instead of a quality product.

In order to insure the accuracy of recipe books provided to the operators, a software package was developed to generate these books from an online system at a semiconductor manufacturing plant. The online system which consists of many pieces of hardware and software, contains up-to-date engineering instructions for each product manufactured at that plant. This software package along with some other hardware and software components is the recipe book generating system that is used to generate recipe books for products that are in their raw state and are ready to be run through the processing cycle. Whenever this system does not function properly, either no product is produced or scrap is generated. Therefore it is very important that problems be resolved immediately so that up-to-date and accurate recipe books can be generated.

The purpose of this project was to develop an expert system that would diagnose the problem when recipe book generating software did not function properly and provide recommendations to resolve the problem. The system questions the user for

symptoms of the problem being experienced and determines cause of the problem by evaluating the user responses to these questions. The final output is an explanation of what is causing the problem and corrective measures that need to be taken to resolve it.

2. *INTRODUCTION*

The objective of this project was to design and implement an expert system that would diagnose problems associated with the malfunctioning of the recipe book generating system. This goal was accomplished by embedding the knowledge of the developer of the software package, which is the major component of this system, into an expert system.

Discussed below are key terms that are referenced frequently in this document and an overview of expert systems.

2.1 KEY TERMS:

The following terms are referenced in this document several times and are defined to help the reader understand this paper thoroughly.

- Front End Processor:

  This term is used for any computer that is connected to another computer which contains a database. When users login to the front end processor, it seems to them that database resides on that computer.

- Database machine:

  Any computer that contains a database. In the context of this paper, Database machine refers to a computer designed and used for database management only. The database machine is transparent to the users because they logon to the front end processor which interacts with the database machine to manipulate data requests.

- Recipe book

  These are books containing step by step instruction for manufacturing a given product in a factory or manufacturing plant.

- Shop Operator:

  A person designated to perform operations on the product as indicated by the recipe book in the manufacturing plant.

2.2 EXPERT SYSTEM OVERVIEW:

An expert systems is a computer system that uses highly specialized knowledge to solve problems at the level of a human expert. In other words, it is a computer system that emulates the decision making ability of an human expert. An expert system basically is composed of the following two parts:

2.2.1 KNOWLEDGE BASE: The Knowledge Base contains facts and knowledge about a specific domain extracted from a human expert. The knowledge can be represented in the knowledge base in several ways such as: frames, network, rules and logical predicates. However, in most of the Expert Systems developed today, knowledge is represented in the knowledge base in the form of rules.

2.2.2 INFERENCE ENGINE: The inference engine manipulates the knowledge contained in the knowledge base to resolve problems posed by the user. It uses the facts and rules embedded in the knowledge base to draw conclusions.

3. LANGUAGE OVERVIEW:

This section gives a brief overview of the Prolog language and Turbo Prolog, the language used to implement this project. It also highlights the commands and features of Turbo Prolog used to implement some of the most important aspects of this project.

Prolog comes from the term Programming in Logic. It is a programming language that uses pattern matching, tree-based data structuring and automatic backtracking for problem solving that involves objects and relations between them.

3.1 DATA OBJECTS:

The Prolog Language supports three types of data objects which are discussed below.

1. *Constants:*

   Constants can be of two different types;1) Atoms and 2) Numbers. The Atoms can be constructed in variety of ways which are described below:

   - Any string consisting of letters, digits, and underscore characters that begins with a lower case letter. For example baby, bill, a21,my_car, etc.

   - Strings composed of special characters. For example <=-->, ..:...:, etc. One has to be careful while using these type of characters because some special character strings are reserved for Prolog and have a predefined meaning. For example ":-" is used as an "if" in Prolog rules.

- Strings of characters enclosed in quotes such as 'Train', 'Water'.

Integers are used to represent numbers so that arithmetic operations can be carried out. Integers are whole numbers consisting of digits only with no decimal points. For example:

0 3 78 231 679329

Real numbers in Prolog can be expressed by using a combination of digits, decimal point and sign. An 'e' is used to indicate an exponent. Listed below are some example of Real numbers.

17

-92

9.82

24 e -14

2. *Variables:*

Variables are strings of letters, digits, and underscore characters that must begin with a capital letter or an underscore sign "-". For example: Book, Output, _LINE, etc.

3. *Structures:*

A structure is composed of several components. It contains a **functor** and one or more **arguments**. The arguments can also be structures. All Structures can be represented as trees. The root of the tree is the functor, and the offsprings of the tree are the components. Following are some examples of structures.

male(name)

father(name,name)

book(title,author)

In the above example, male, father, and book are functors while name, title, and author are arguments.

Now that we have discussed the basic data objects in Prolog, the next section will describe how Prolog uses these data objects to describe objects and relationships between them.

3.2 FACT:

In Prolog a *fact* describes objects and the relationship between these objects. In other words, facts in Prolog allow one to express arbitrary relationships between objects. Consider the following fact (structure).

likes(bill,nancy).

This fact has two objects, bill and nancy and a relationship called likes. This fact simply states that object "bill" likes the object "nancy". A simple Prolog program could contain one or more such facts. In each fact, the object outside the parenthesis is called **predicate** and the objects inside the parenthesis are called **arguments**. A collection of facts is referred to as a database in Prolog.

3.3 GOAL:

Once facts are present, questions regarding these facts can be asked. Questions are referred to as *goals* in Prolog. A goal in Prolog looks like a fact, except a special symbol is put in front of it. When a goal is presented to Prolog, it will search through the database to satisfy this goal. It tries to match the fact in the goal to the facts present in the database.

Two facts will match if their predicates are the same and they both have the same arguments. If Prolog finds a fact that matches the goal, it will respond with a "yes", otherwise it responds with a "no". Consider the following database.

```
likes(bill,oranges)
likes(bill,nancy)
likes(nancy,movie)
likes(bill,movie)
```

Now if we ask Prolog

```
?- likes(bill,nancy)
```

It will respond with a "yes" because there is a matching fact in the database. If the following goal is presented to Prolog

```
?-likes(bill,cake)
```

It will respond with a "no" which implies that it could not find a matching fact in the database. In order to find out what objects a given object likes, we will have to use

variables. For example in order to find out what bill likes, the goal would be presented in the following format.

?-likes(bill,X)

To satisfy the above rule, Prolog would search the database for a fact in which matches "likes" and whose first argument is "bill". Then it uses X to represent all things that bill likes. So the response would be

X=oranges

X=nancy

X=movie

3.4 RULE:

In order to make a simple Prolog program sophisticated, *rules* can be added to the database. A rule is a general statement about objects and their relationships. In other words, a rule is a type of a fact that depends on a group of facts. It is true whenever a particular condition is satisfied. An example of a rule is given below.

likes(john,X):-likes(X,movies).

The above rule states that john likes anybody who likes movies. The rule has two parts: the condition to be satisfied on the right-hand side, and the conclusion on the left-hand side. A rule can have more than one condition on the right-hand side. If these conditions are separated by commas, then all of them will have to be satisfied at the same time in order for the left-hand side to be true. But if these conditions are separated by semicolons, then satisfying any one of them would make the

conclusion true. Consider the following example.

```
likes(john,X):-
        likes(X,movies),
        likes(X,books),
        male(X).
```

The above rule states that john likes anybody who likes movies and books and is male. Since the conditions on the right-hand side are separated by commas, a person would have to satisfy all three in order for john to like them. Now consider the following rule.

```
big(Y):- huge(Y);
         tall(Y),
         large(Y);
```

In this rule Y is big if any of the three conditions on the right hand side is satisfied. All three condition do not have to be satisfied at the same time.

3.5 TURBO PROLOG®:

Turbo Prolog is an implementation of Prolog that can be used for expert system development. It provides a full screen editor, along with multiple window facilities and interactive debugging. Turbo Prolog is a compiled language, therefore it is faster than other implementations of Prolog which are interpreted. In the following section the program format required by Turbo Prolog will be discussed along with the commands and features used in the development of this project.

---

0. TURBO PROLOG is a Registered Trademark of Borland International.

A Turbo Prolog program consists of five sections which are the domain section, the database section, the predicates section, the goal section, and the clauses section. Each section is briefly discussed below.

- The domain section is the first section of a Turbo Prolog program and begins with the keyword "domain". It contains the declaration of domain, which describes the different classes of objects used in the program. For example:

```
person = symbol

country = symbol

conditions = category *

category = symbol

ruleno,condno,fileno = integer
```

- The database section contains the predicates used in the dynamic database. A program that doesn't need a database would not contain this section. Below is an example of a predicate that would be in the database.

```
rule(ruleno,category,category,conditions)
```

- The predicate section contains all definitions of all the predicates used in the program. For example:

```
member(category,conditions)

border(country,country)
```

```
female(person)

parents(person,person)
```

- The goal section states the goal that the program has to satisfy. There are two types of goal; internal and external. A program with an external goal is interactive and missing the entire goal section. Following are some example of goals presented to a program.

```
female(Y)

border("Spain",X)

parents("Bill",X)
```

- The clauses section contains all the facts and rules in the program. For example:

```
female("Nancy").

border("Spain","Italy").

parents("Bill","Mary").

mother(X,Y):-
        parent(X,Y),
        female(X).
```

The following section contains a brief description of Turbo Prolog commands used in this project.

1. *makewindow* – is used for windowing operations. It enables the user to create new windows by specifying the number, size, height, length and color of the window. The number assigned to each window is referenced by other Turbo Prolog predicates such as *gotowindow*. The makewindow predicate was used in this project to create the user interface which consists of a full screen window that is used to present and receive information from the user.

2. *clearwindow* – is used to clear any text or graphic displays from the current window. This predicate was used to clear the screen before a new session was started with the Expert System.

3. *existfile* – is used to verify if the given DOS files exist. This command in conjunction with the *consult* predicate was used to verify the existence of the database file.

4. *consult* – is a database predicate that is used to load the specified database file into memory. This predicate fails if the database file doesn't exist or if not enough memory is available or if predicates in the database file do not agree with database predicate declarations in the program.

5. *str_int* – is used for converting string representation of integer numbers to variables of type integer. This predicate converted the condition numbers specified in the rule from string to numeric.

6. *upper_lower* – is used to convert text to lower or upper case. This predicate was used to convert all user response to upper case to limit the amount of error checking that needed to be done.

7. *assert* – is used to insert facts into the database. This predicate was used to store the user responses in the database.

8. *retract* – is used for removing a specified clause from the dynamic database. A dynamic database only consists of facts. This command was used to remove all previously stored user responses from the database before a new session was started with the Expert System.

9. *readln* – is used to read character string input from the current input device.

10. *frontchar* – Splits a character string by separating the first character from the rest of the string. It puts the new values into the specified variables. The original string stays intact. This predicate was used to separate the condition number from the expected user response for that condition as applicable to the rule that contained it.

4. PROBLEM DESCRIPTION:

This section describes the function and design of the recipe book generating system. This system consists of recipe book generating software, a computer which is used as a front end processor, a database that contains up-to-date manufacturing instructions for a given line of product, process control software that download recipes to the database and the printers that actually print the recipe books and the network which connects all hardware components together.

The recipe book generating software package resides on a front end processor that is connected to a database machine via a network. The database machine contains multiple databases, each of which contain recipes for a given line of product. The printers are directly connected to the front end processor. This front end processor also contains the process control software.

The users (shop operator) access the system (front end processor) via terminals located in various areas of the manufacturing plant. When user invokes the recipe book generating software, it prompts for several things such as the product name, the exact recipe name, name of the first operation in the recipe, name of the last operation in the recipe, number of recipe books to be generated, type of recipe books (whole or partial).

This software package performs extensive error checking before accepting the input from the user. It validates all input data against the database and reports any inconsistencies to the user and requires the data to be corrected.

After validating the information received from the user, it retrieves the data from the designated database and formats the recipe books. After formatting the recipe books, it queues them for printing.

This process of generating recipe books using this system seems very easy and straight forward until incorrect or incomplete books are generated by the system. Since the user is not familiar with this recipe book generating system beyond the point of requesting books, he/she has no idea what has gone wrong. Depending on the problem being experienced, there are many variables that could influence this process and all such variables are unknown to the user. The following section discusses the types of problems experienced by the users while generating recipe books.

4.1 PROBLEM TYPES:

Problems that can occur while generating recipe books can be categorized into four different categories based on the symptoms experienced by the user. The categories are 1) No Books 2) Incomplete Books 3) Incorrect Books and 4) Printer jamming while printing the recipe books. These categories are discussed in detail in this section.

4.1.1 NO BOOKS:

This category applies when user does not receive any recipe books in a specified time frame after submitting a request. Some reasons for recipe books not being printed could be that the database is locked, the network is down, the front end processor runs out of disk space, the line spooler for printing is not functional, the

physical connections between the printers and front end being loose, etc.

4.1.2 INCOMPLETE BOOKS:

This category is applicable when the user is getting incomplete recipe books. In other words, the recipe books generated do not contain complete instructions to manufacture the product. Some of the reasons for incomplete recipe books are missing critical links between data item, corrupted data, the user choosing the wrong option from the menu.

4.1.3 INCORRECT BOOKS:

This category refers to incorrect books that might be generated due to a variety of reasons such as; user entering data out of sequence, corrupted data links, etc.

4.1.4 PRINTER JAMMING:

The last category consists of problems associated with printing the recipe books. Problems belonging to this category are relatively easier to debug because there are limited variables that can effect this phase of recipe book generation. we already know that a book was generated because the printer is trying to print it. In other words, the problem is either the printer or the data it is trying to print.

5. PROGRAM DESCRIPTION:

This section will describe in detail the actual Expert System that was developed to trouble shoot recipe book generating software used in a manufacturing plant.

5.1 PURPOSE:

This Expert System encompasses the relevant knowledge of the developer of recipe book generating software. It's main purpose is to diagnose problems based on user input and suggest corrective action.

5.2 INPUT:

This diagnostic expert system requires the following input from the user to trouble shoot the system.

5.2.1 PROBLEM DEPENDENT INPUT:

This type of input depends on the type and nature of the problem. The Expert System asks the user different questions depending upon the problem he/she is experiencing. Users has to answer these questions either by selecting from a menu displayed by the Expert System or by responding with a yes/no.

5.2.2 PROBLEM INDEPENDENT INPUT:

This type of input is required regardless of the problem. In other words, users has to answer the following questions no matter what the problem is.

- The system/database being used for generating recipe books when the problem

occurred.

- The type of problem occurring with the recipe book generating software.

5.3 OUTPUT:

The output explains the cause of the problem and suggests corrective action.

5.4 LANGUAGE:

This Expert System was developed in TURBO PROLOG. The reasons for choosing Turbo Prolog are following.

- Requires much less code to solve the problem than other languages that I know.
- Its power of searching, pattern matching and backward chaining.
- The ability to manipulate internal data as well as external data (from a file) in the same manner. Does not require special code to read & write to and from a file.

5.5 MACHINE IMPLEMENTATION:

This Expert System will reside on IBM compatible personal computer running MS-DOS and Turbo Prolog.

5.6 KNOWLEDGE REPRESENTATION:

The knowledge base is a collection of production rules that are linked together by the inference engine. All information is embedded in these rules. The format of the rules is as follows:

```
rule(number,description,action,conditions)
```

cond(cond_number,question)

how(cond_number,instruction)

why(cond_number,explanation)

errfiles(error_type,error_number,error_text)

action(error_type,error_number,corrective_procedure)

Examples:

rule(1,"action","start the line spooler on your system",["y1","n2"])

rule(2,"action","bring up the network",["n1","y3","n4"])

cond(1,"Do you see any print request on the printer queue")

cond(2,"Is the line spooler running")

cond(3,"Do you see the create_book program running")

cond(4,"can you access the database machine")

how(1,"execute the command lpstat -o")

why(1,"If there are requests on the print queue,then the problem
is between the front end processor and the printers. The books
have been created and are waiting to be printed").

errfiles("LOG",1,"Cannot open datafile in /tmp directory.")

action("LOG",1,"Check permissions on the /tmp directory and
change them to a 777")

Explanation:

rule:

first field: rule number

second field: descriptive field

third field: corrective action to be taken if all the conditions are met.

fourth field: list of conditions that have to be satisfied in order to fire this rule.

cond:

first field: condition number

second field: question that user has to answer

why:

first field: number of the condition to which this rule applies.

second field: explanation as to why the user is being asked that question.

how

first field: the condition number to which this instruction applies.

second field: instruction to perform the action.

errfiles:

first field: type of error message. This indicates which log file error message appeared in.

second field: specifies the error number

third field: displays the actual text of the error message.

action:

first field: type of error message. (same as in errfiles)

second field: specifies the error number (same as in errfiles)

third field: corrective action to be taken

5.7 INFERENCE ENGINE:

The inference engine consists of operating rules and principles. It uses knowledge base rules to make decisions. Following are the steps that are followed to produce the final output. Examples are given to specify how inference engine reaches a certain conclusion.

a. The Expert System starts off by searching the knowledge base for system/database names that use the recipe book generating system. This list

of system/database is presented to the user who selects the system/database that he/she is experiencing the problems with. The knowledge base contains such information as how many different databases reside on a system, number of printers setup on the system as well as the printer configuration. This information is used in situations where recipe books are being created on a system and queued for printing but the user does not see them being printed. For example if the user is not getting books on printer C from system A and recipe books are being created and queued for printing, the problem might be that printer C is not configured on system A and the user doesn't know this fact.

b. Then a list of possible types of problems that could be experienced with recipe book generating system is displayed to the user who selects the particular problem he/she is experiencing.

c. After user specifies the problem he/she is experiencing, the Expert System searches the knowledge base and tries to find all applicable rules for that particular problem by pattern matching. It starts with the first applicable rule and if this rule fails, it goes to next rule and keeps on trying until a rule is successful or it runs out of rules to apply.

d. By matching the rule, the Expert System determines what questions to ask the user. It takes the conditions associated with that rule and presents them to the user as questions. If all the conditions associated with that rule are met, then that rule is fired and the user is presented with diagnoses of the problem along with the recommended solution. Following is an example that would illustrate how the Expert System will determine if a condition has been met.

Example:

Let's assume that following is the first applicable rule in the knowledge base.

```
rule(1,"NO_BOOKS","The create_book program is waiting for space
 to become available on the front end processor. Make sure
   the space in /usr is above 5000 blocks.", ["n1","y3","y6"])

cond(1,"Do you see any print requests on the print queue")

cond(3,"Do you see the create_book program running")

cond(5,"Is the create_book program gaining time")

cond(6,"Do you see any error messages in the log file")
```

In the above example if user is not getting any recipe books at all, the Expert System chooses rule 1 that is applicable to NO_BOOKS category and starts processing the conditions associated with this rule. It takes "n1" and breaks it into two parts "n" and "1". "1" represents the condition or question that the user has to answer and "n" represents the answer that is expected for this condition to be satisfied for this particular rule. Another method to represent the above rule is as follows.

```
If problem(NO_BOOKS) and

no books are queued up for printing and

create_book program is running and
```

create_book is not gaining time and

error messages appear in the log file

THEN tell the user

create_book program waiting for space on the front end

processor and to make sure space is above 5000 blocks in

/usr file system.

e. If the user specifies that an error message is being written to the log file, a list of error messages that would support the current rule and could appear in the log file is displayed. If an error message is from the displayed list of errors, then the rule is fired otherwise another rule is invoked.

f. There are several rules for a given category such as NO_BOOKS and each rule has some conditions associated with it that have to be satisfied before the rule could be fired. All rules have the same conditions associated with them but these conditions appear in different combinations and are satisfied by different responses. Consider the following example.

rule(1,"NO_BOOKS","Bring up the network",["n1","y3","n5"])

rule(2,"NO_BOOKS","Clean up the space in /usr",["n1,"y3","y5","y6"])

rule(3,"NO_BOOKS","Make sure the printer is online",["y1","y2"])

In the above example all three rules use conditions 1-6 in different combinations. Rule 1 and 2 both use conditions 1, 3 and 5 but in rule 1,

condition 5 will be satisfied if user responds with a "no" while in case of rule 2, condition 5 is satisfied if user responds with a "yes". If the user response to condition 5 was a "yes", the Expert System would not only abandon rule 1 but any rule that requires condition 5 to be false. It would proceed to process rule 2. The Expert System goes through all NO_BOOK category rules one at a time until all conditions for a particular rule are satisfied and a conclusion can be drawn or all rules for that problem are exhausted.

g. All user responses are stored in the database in the following form:

yes(1)

no(1)

where 1 is the condition presented to the user and "yes/no" are user responses.

The Expert System uses the stored user response to determine which rules to eliminate from the list of applicable rules. For example, if user responded with a "no" for condition 1, then no(1) would be stored in the database and when the Expert System is going through the applicable rules, it will search the database for yes(1) and no(1). Once it finds no(1), it will eliminate all rules that require condition 1 to be true.

Another purpose for storing the user response is to keep track of all questions already posed to the user. Consider the following example.

rule(7,"INCOMPLETE_BOOKS",["y5","n6"])

rule(8,"INCOMPLETE_BOOKS",["y5","n7"])

In the above example, when rule 7 is being processed and the user responds with a "yes" to condition 5. This response is stored in the database as yes(5). Let's assume that the user responds with a "yes" for condition 6, rule 7 is abandoned at this point because this rule requires this condition to be false. Now lets assume that the Expert System will select rule 8 for processing. It will check the database to see if the user was asked this question before. It finds yes(5) in the database which implies that this question already has been asked and condition 5 is true so it can keep processing this rule until it finds a condition that is not satisfied by the user response.

h. The expert system provides two help facilities to the user. The first facility allows the user to question why he/she is being asked a particular question by typing in "why" when prompted for a response to a question. The Expert system knowing the condition number searches the knowledge base for predicate "why" that applies to the given condition number. On finding the explanation it displays the explanation to the user otherwise it informs the user that it cannot find any explanation in the knowledge base.

i. The second facility is the "how" facility that allows the user to ask the Expert System how to perform a certain action. This facility is intended to be used when the user cannot answer a certain question because he/she doesn't know how to get that information. For example if Expert System asks the user if there are any print requests on the print queue, the user might not know how to check if there are any print requests queued for the printer. At this point if "how" is entered at the prompt the Expert System will display the following information to the user.

Type "lpstat -o" at the prompt and hit the enter key.
if the prompt comes back with no information then that implies
that there no requests queued up.

5.8 SUMMARY:

The Expert System greets a user with a menu that displays all the words in it's vocabulary which are following:

1. yes
2. no
3. how
4. why
5. exit

Following are some of the features of this expert system:

- It converts all user input to capital letters so that it has to do less error checking.
- If the user responds by entering a word not in it's vocabulary, it generates an error message and prompts the user again.
- It allows the user to ask for the reason for asking a certain question by typing "why" at the prompt. If the Expert System cannot find the reason in the knowledge base, it notifies the user.
- The user can also ask the Expert System how to perform a certain action. If it cannot find the answer in the database, it will notify the user.

- It allows the user to end the session any time by simply typing in exit.
- After a session is completed, it asks the user if he/she would like to initiate another session. It starts another session if user responds with a "yes", otherwise it terminates.

This expert system is application independent since it is logic based. The knowledge base is totally separate from the inference engine and is loaded from an external file. It can be used for any diagnostic application as long as the user conforms to the data format of the rules. If conditions are such that no rule can be fired, it suggests to the user to contact the local support. The system relies on the user input, it has no way to verify if that condition actually exist or user has made a mistake. The user is only allowed to respond with a *yes/no* to minimize the problem of uncertain and incomplete data.

5.9 MANAGEMENT OF UNCERTAINTY:

The application implemented in this project did not require managing uncertain or incomplete data because the user is expected to have enough information to be able to respond to a question with a yes/no.

5.10 SAMPLE SESSION:

Following is an example of a session with the Expert System. In this example, the user has submitted a request to the recipe book generating system and is receiving incomplete books. He starts a session with the Expert System to determine what is causing the recipe book generating software to create incomplete recipe books.

The user starts up a session by typing DES (Diagnostic Expert System) at the

prompt. System responds in the following manner.

```
**************************************************
*                                                *
*      Welcome to the Diagnostic Expert System   *
*                                                *
*      Please respond by putting 'yes' or 'no'   *
*      or selecting a number from the menu.      *
*      You can also use 'why' or 'how'.          *
*      Type 'exit' to end the session at any time.*
*                                                *
**************************************************
```

Loading the Knowledge Base. Please wait ....

DES: Following are the systems/databases in my knowledge base. Please select the system/database where you are experiencing the problem.

1. System A/Product I
2. System A/Product_G
3. System B/Product_K
4. System B/Product_P

Enter your selection: 2

DES: Following are the problem categories in my knowledge base. Please select the problem you are experiencing.

1. No books being printed.
2. Incomplete books.
3. Incorrect books.
4. Printer is jamming or garbling the books.

Enter your selection: 2

DES: Does each incomplete book have a cover page? yes

DES: Are the cover pages correct? yes

DES: Are the cover pages complete? yes

DES: Are you getting incomplete books for all codes? yes

DES: Do all codes have any data items common? yes

DES: Do all incomplete books stop at the same data item? yes

DES: Does the data item where all incomplete books stop, exist in all codes? yes

DES: Was there a download of this item processed recently (2-3 days)? how

DES: Login to the download system, select the search option and enter the data item name at the prompt. When data item is displayed, press the F9 key and looked at the most recent date in the left most column of the screen.

DES: Was there a download of this item processed recently(2-3 days)? yes

DES: Have you received any complete recipe books for any code since the download? no

DES: It seems that data item (at which all incomplete books stop at) is not linked to the next data item of the recipe. Login to the download system, go into the browse menu and enter the last data item (from the incomplete books) at the prompt and check it's links. If any links are

missing, use the "estab_link" command to establish the missing link and resubmit your request.

CONCLUSION

The Diagnostic Expert System for trouble shooting the recipe book generating system was successfully implemented in Turbo Prolog. All known possible types of problems that can occur while using the recipe book generating system were embedded in the database along with their causes. The Knowledge base also contained the conditions/steps that inference engine used to determined cause of the problem. The "how" and "why" facilities which enable the user to ask the Expert System for instructions to perform certain tasks as well as able to question it's reasoning for posing a certain question make this Expert System very user friendly.

This Expert System will maximize the benefits of recipe book generating system by enabling the users to resolve recipe book generating system problems efficiently and effectively. It emulates the human expert by applying a structured approach to problem solving by asking the user questions in a logical order. This logic is built in the knowledge base rules which are setup such that more probable and important causes are explored first.

The overall purpose of this project was to capture the expertise of the developer of recipe book generating software which is the major component of the system into an Expert System. As this document indicates, the objective of this project was successfully accomplished and the Expert System developed can successfully assist the users in resolving recipe book generating system problems efficiently as the human expert.

REFERENCES

Bratko, Ivan, Prolog Programming for Artificial Intelligence,1986 Addison-Wesley Publishers Limited.

Clocksin, F. W. and Mellish, S. C., Programming in Prolog. New York Tokyo Springer-Verlag Berlin Heidlberg, 1984

Yin, Maung Khin and Solomon, David, Using Turbo Prolog. Indiana, Indianapolis: Que Corporation,1987.

VITA

- Name:

  Shahida Mahmood Parvez

- Parents:

  Raja M. Ashraf

  Anar B. Ashraf

- Date of Birth:

  April 14, 1963 Jhelum, Pakistan

- Education:

  Bachelor of Arts in Computer Science, 1985

  Hunter College, CUNY, New York

  Master of Science in Computer Science, 1992

  Lehigh University, Pennsylvania

- Honors:

  Magna Cum Laude

- Professional Experience:

  Computer Engineer 1985 - 1990

  AT&T Microelectronics

  Reading, Pennsylvania

  Software Engineer (Current Employment)

  AT&T Information Management Systems

  Piscataway, New Jersey

END

OF

TITLE