An expert system for the diagnosis of vehicle malfunctions.

Selek, Can.

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN EXPERT SYSTEM FOR THE DIAGNOSIS OF VEHICLE MALFUNCTIONS

by

Can Selek

December 1987

Thesis Advisor Neil C. Rowe

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION: Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School; 6b OFFICE SYMBOL (If applicable): 52	7a NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code): Monterey California 93943-5000	7b ADDRESS (City, State, and ZIP Code): Monterey California 93943-5000
8a NAME OF FUNDING SPONSORING ORGANIZATION; 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State and ZIP Code)	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
An Expert System for the Diagnosis of vehicle Malfunctions

12 PERSONAL AUTHOR(S)
Can Selek

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year Month Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	December 1987	76

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Expert System Diagnosis, Prolog

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

We examine the feasibility of an expert system to assist in the diagnosis of vehicle malfunctions. A passive expert planner is proposed that utilizes multiple domain-dependent knowledge bases. The system is implemented on a personnel computer, and is based on general-purpose car repair manuals. An efford is made to adequately define emergencies. The knowledge base and inference procedure for such a system are also presented.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION: Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL: Neil C. Rowe	22b TELEPHONE (Include Area Code): (408) 646-2462; 22c OFFICE SYMBOL: Code 52RP

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

U.S. Government Printing Office 1986—606-243

Approved for public release; distribution is unlimited

An Expert System for the Diagnosis of Vehicle Malfunctions

by

Mufit Can Selek
Lieutenant Junior Grade, Turkish Navy
B.S., Turkish Naval Academy, 1981

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1987

ABSTRACT

We examine the feasibility of an expert system to assist in the diagnosis of vehicle malfunctions. A passive expert planner is proposed that utilizes multiple domain-dependent knowledge bases. The system is implemented on a personal computer, and is based on general-purpose car repair manuals. An effort is made to quantify the amount of information processing necessary to adequately define the problem. The knowledge base and inference procedures for such a system are also presented.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. BACKGROUND

The goal of artificial intelligence (AI) is to develop computer programs that could in some sense think; that is, solve problems in a way that would be considered intelligent if done by a human.

Artificial intelligence imitates both the basic problem solving and learning process of human beings. "In the sixties, AI scientists tried to simulate the complicated process of thinking by finding general methods for solving broad classes of problems; they used these methods in general-purpose programs. However, despite some interesting progress, this strategy produced no breakthroughs. Developing general-purpose programs was too difficult and ultimately fruitless." [Ref. 1]

"It wasn't until the late 1970s that AI scientists, began to realize something quite important: the problem-solving power of a program comes from knowledge it possesses, not just from the formalisms and inference schemes it employs." [Ref. 1] This realization led to the development of special-purpose computer programs, systems that were expert in some narrow problem area. These programs were called **expert systems**. This knowledge can

often be accessed much faster and with greater accuracy in a computer than from the human expert.

Within the last few years, research in the field of artificial intelligence has grown significantly. Early pioneers of expert systems, such as MYCIN [Ref. 2] in the area of medical diagnosis, have proven successful enough to motivate the investigation of similar expert systems in others fields of study. One such product related to vehicle diagnosis, besides equipment maintenance expert systems in commercial and industrial environments, has recently been put into service by ANALYTICS; it is called AIJPA (Artificial Intelligence Job Performance Aid).

B. OVERVIEW

While vehicle repairs can be made by many people, accurate troubleshooting is a rare skill for the amateur and the professional alike. For that reason, a shortage of technical experts could have a drastic effect on the efficiency of equipment malfunction diagnosis. The personnel performing this task must possess both a good technical background and a great deal of training and experience on the particular equipment being repaired. Therefore, an expert system applied to vehicle diagnosis could improve job performance, with less maintenance cost. It will also help solve the lack of well-trained professionals.

Thus, the purpose of this thesis is to assess the feasibility of augmenting the mechanic with a computer in the maintenance of vehicle. Based on current diagnostic expert systems such as MYCIN [Ref. 2], our expert diagnosis system will contain a knowledge base obtained from technical manuals and expert mechanics. It should be possible for less-qualified mechanics to quickly and accurately assess vehicle malfunctions through interaction with the expert system.

The particular objectives are as follows:

1. Quantify, through the implementation, the amount of information processing necessary to sufficiently diagnose the vehicle malfunctions.

2. Evaluate the problems encountered in implementation when a rule-based expert system is chosen as a basis.

3. As far as the effectiveness of programming languages is concerned, evaluate the efficiency and ease of use of PROLOG throughout the implementation.

The implementation of a small-scale expert diagnosis system demonstrates the feasibility of a larger-scale system. This implementation involves several sections of the engine system of the vehicle. The design is implemented in Arity Prolog [Ref. 3] on an IBM XT computer.

Three assumptions have been made to simplify the implementation.

1. There is no partial certainty assigned to the existence of any individual malfunctions.

2. All indications presented to the mechanic by test equipment are correct.

3. There exists at least one malfunction.

II. BACKGROUND

A. EXPERT SYSTEM CONCEPTS

An expert system is an extensive body of knowledge about a specific problem domain. Characteristically, this knowledge incorporates facts and rules from human experts and written documentation, and is made available to users, applying some inference mechanism which governs those rules, to provide solutions to the problems brought by its users.

1. Features of Expert Systems

As Forsyth [Ref. 4] lists some of the distinctive features of expert systems:

a. An expert system is limited to a specific domain of expertise.

b. It can reason with uncertain data.

c. It can explain its reasoning path in a comprehensive way.

d. The facts and inference mechanism are clearly separated. (Knowledge is not encapsulated into the deductive procedures.)

e. It is designed to grow incrementally.

f. It is typically rule-based.

2. Building an Expert System

"The main players in the expert system environment are the expert system, the domain expert, the knowledge

engineer, the expert-system-building tool, and the end user" [Ref. 1]. Their basic roles and their relationship to each other are summarized below.

a. The Domain Expert: A person who, through years of training and experience, has become extremely proficient at problem solving in a particular domain.

b. The Knowledge Engineer: A person who, with a background in computer science, knows how to build expert systems. The knowledge engineer interviews the domain expert, organizes the knowledge, and decides how it should be represented in the expert system.

c. The Expert-System-Building Tool: Both the programming language and support environment used by the knowledge engineer or programmers to build the expert system.

d. End User: The person for whom the expert system was developed.

3. Expert System Architecture

a. Knowledge Organization

Expert systems need to be organized in an orderly manner to avoid confussion. In general, the knowledge is divided into three categories. [Ref. 4]

1. Factual Knowledge: This knowledge represents a particular case and is usually gathered through a dialogue with the user.
2. Procedural Rule Knowledge: This knowledge is usually collected in advance from the domain specialist and forms the core of a knowledge base. This also forms the reasoning part of the system to infer conclusions.
3. Control Knowledge: The system needs to have a variety of control strategies available to it so that alternatives can be tried out at run time.

b. Procedural Knowledge

The procedural knowledge most commonly used in current expert systems is a rule-based knowledge representation in the form of IF <condition> THEN <action> statements, as shown in Figure 2.1.

IF

1. STARTER TURNS BUT NOT THE ENGINE, and
2. NO HIGH RESISTANCE TO MOVING ENGINE

THEN

CONCLUDE THAT THERE IS A MALFUNCTION IN STARTER UNIT

Figure 2.1 Sample IF ...THEN ... INFERENCE RULE

When the current problem situation satisfies or matches the IF part of a rule, the action specified by the THEN part of the rule is performed. This matching of rule IF portions to the facts can produce what are called **rule firings** or **inference chains**.

c. Control Knowledge

The inference engine reasons and makes inferences based upon the application of rules and facts contained in the knowledge base. It accomplishes this through a control structure. There are many control structures implemented in current rule-based expert systems.

One is known as **backward chaining** or **goal-directed** reasoning. This structure begins with the selection of a specific goal and then searches the rules to find those whose concequent actions can achieve that goal. Backward chaining is often a good control structure when there are more facts than final conclusions (goals). [Ref. 5]

Often, rule-based systems work from just a few facts but are capable of reaching many possible conclusions [Ref. 5]. For that reason, it makes more sense to match the rules to the state or the condition of facts and continually apply those rules to new states until the desired goal

is attained. This control structure is known as **forward chaining** or **data-directed computation**. Different control structure ideas can be combined in **hybrid** control structures. "Hybrids of forward and backward chaining, compromising on the advantages and disadvantages of both, are often used. The most common is the **rule-cycle-hybrid** control structure because it is easy to implement. With rule-cycle-hybrid, rules are tried in order as with backward chaining, but each rule is used in a forward-chaining way to assert new facts." [Ref. 5]

d. Factual Knowledge

A knowledge base contains facts, assertions, and rules. Some of the facts are short-term information that can change rapidly during the course of a consultation. Facts in a data base are normally passive; they are either there or not there. A knowledge base, on the other hand, actively tries to fill in the missing information [Ref. 4].

e. User Interface

Whatever the style of expert systems, they assume that information is provided manually through a question asking/answering dialogue, or automatically by means of sensors or other

devices. The interface is that part of the expert system which controls communication with the user.

B. SUMMARY OF CURRENT KNOWLEDGE IN DIAGNOSIS

1. Theory of Diagnosis

"In the theory and design of diagnostic reasoning systems, there appear to be two quite different approaches in the literature. In the first approach, often referred to as diagnosis from first principles, one begins with a description of some system with an observation of the system's behavior. If this observation conflicts with the way the system is meant to behave, one is confronted with a diagnostic problem, namely, to determine those system components which, when assumed to be functioning abnormally, will explain the discrepancy between the observed and correct system behavior. For solving this diagnostic problem from first principles, the only available information is the system description, i.e., its design or structure, together with the observation of the system behavior. In particular, no heuristic information about system failures is available, for example, of the kind **when the system exhibits such and such aberrant behavior, then in 90 percent of these cases, such and such components have failed**." [Ref. 6] An example is electronic equipment troubleshooting.

"Under the second approach to diagnostic reasoning, which might be described as the experiential approach, heuristic information plays a dominant role. The corresponding diagnostic reasoning systems attempt to codify the rules of thumb, statistical intuitions, and past experience of human diagnosticians considered experts in some particular task domain. The structure or design of the corresponding real-world system being diagnosed is only weakly represented, if at all. Successful diagnosis stem from the codified experience of the human expert being modeled, rather than from what is often referred to as **deep** knowledge of the system being diagnosed." [Ref. 6] A notable example is the MYCIN system.

2. The Behavior of the Diagnostic Reasoning Task

"Engineers and scientists constantly strive to understand the differences between physical systems and their models. Engineers troubleshoot mechanical systems or electrical circuits to find broken parts. Many everyday common-sense reasoning tasks involve finding the difference between models and reality." [Ref. 7]

Diagnostic reasoning requires a means of assigning credit or blame to parts of the model based on observed behavioral discrepancies. If the task is troubleshooting then the model is presumed to be correct and all model system differences indicate part malfunctions, as shown in Figure 2.2. [Ref. 7]

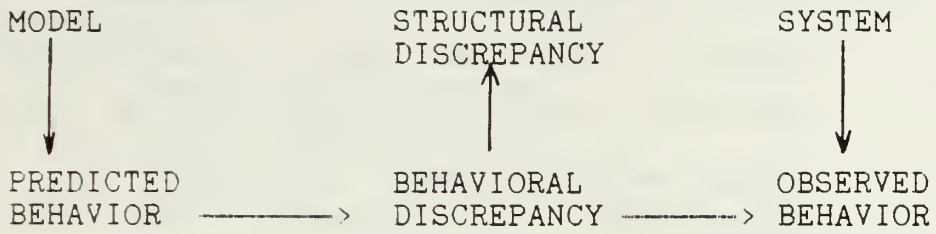

Figure 2.2 Model-System Difference

Usually, the initial evidence does not imply a unique explanation. Then the diagnosis requires two phases. The first, mentioned above, identifies the set of possible model-system differences. The second proposes evidence-gathering tests to refine the set of possible model-sytem differences until they accurately reflect the actual differences. [Ref. 7]

"When these theoretical principles are applied in practice, it is important to note that experts seldom solve problems using a rigorous theoretical analysis; rather, their understanding has a more ad hoc character. Some excellent troubleshooting technicians, for instance, use very little theory. Yet, by using their own simpler **conceptual models**, they can troubleshoot a faulty device quite efficiently." [Ref. 8]

The thought process of a typical human expert for an engine under diagnosis is:

a. Observe the behavioral discrepancies.

b. Identify the subsections of the engine system which may explain the behavioral discrepancies, using the structural definitions of system model.

c. Starting from the most problem-related subsection, search a known fault data base within that subsection, to figure out the relevant faults using heuristic information.

d. Find a method supported by procedural information to test and prune the possible faults.

3. Concept of the Diagnosis of Vehicle Malfunctions

The familiar example of a car engine system is chosen as a model for the implementation of a small-scale system which will diagnose vehicle malfunctions.

To gather information pertaining to procedural knowledge from sources and convert them into facts and rules format, it is essential to study car repair guides which are a good source of information for how knowledge should be represented. An engine system is composed of five major subsections [Refs. 9, 10]:

1. Battery section.
2. Cranking/Starter section.
3. Ignition section.
4. Fuel section.
5. Compression section.

And engine malfunctions can be separated into two general groups:

1. Engine will not start.

2. Engine performs poorly.

A common approach in the troubleshooting sections of car repair guides is to use charts of the most common symptoms of engine malfunctions. For each symptom, relevant subsystems of the engine system are indexed as in Figure 2.3.

PROBLEM: SYMPTOM	BEGIN AT SPECIFIC SUBSYSTEM
Engine Will Not Start	
Starter does not turn	Battery, Starter
Starter turns, engine doesn't	Starter
Starter turns engine very slowly	Battery, Starter
Engine fires intermittently	Ignition
Engine Performs Poorly	
Hard Starting	Ignition, Fuel
Rough idle	Ignition, Fuel
Backfire through the carburetor	Fuel, Ignition, Compression
Backfire through the exhaust	Fuel, Ignition, Compression
Blue exhaust gases	Compression
Black exhaust gases	Fuel

Figure 2.3 Index to Problems

The user is expected to proceed to an indexed subsystem. Sections are arranged so that following each

test, instructions are given to proceed to another, until a problem is diagnosed. For instance, in the sample for ignition section shown in Figure 2.4:

TEST AND PROCEDURE		RESULTS AND INDICATIONS	PROCEED TO
4.1 Check for spark Hold each spark plug wire approximately 1/4" from ground with gloves or a heavy, dry rag. Crank the engine and observe the spark.	---->	If no spark is evident	4.2
	---->	If spark is good in some cases	4.3
	---->	If spark is good all	4.6

Figure 2.4 Sample Section

This type of approach shows a decision lattice representation, a strictly procedural nature. However, decision lattices are better for troubleshooting manuals than for computers. Despite some of the benefits of decision lattices in terms of ease of implementation and the minimal numbers of questions necessary to establish conclusions, they shouldn't be considered as general structure for an expert system due to some of the following disadvantages as stated by Rowe [Ref. 5];

1. They can't always reason efficiently.
2. They are difficult to modify.
3. They may be difficult to design, since at each point we have to determine the best question to ask.

Also, a human expert searching a known fault set for that subsystem uses heuristic information, and then tests and prunes until some is found, not a procedural method.

III. DESIGN AND IMPLEMENTATION

This chapter describes what techniques were used to construct the expert system and why this particular implementation was chosen.

A. PROGRAMMING LANGUAGE

The computer programming language **Prolog** is quickly gaining popularity throughout the world, in artificial intelligence applications. Prolog is like logic in that it can infer solutions to problems. Prolog's ability to infer solutions to problems changes the way in which programmers work.

"Prolog has three positive features that give it key advantages over conventional programming languages. First, Prolog in syntax and semantics is so close to formal logic that programs are better understood and better maintained. Second, Prolog provides automatic backtracking, a feature that simplifies searching alternatives. Third, Prolog allows a procedure definition to be used for many different kinds of reasoning." [Ref. 5]

The availability of a Prolog programming tool for the IBM XT was a very important factor in this implementation. The design was implemented with the Arity/Prolog interpreter

which is a product of Arity Corporation, designed to be a powerful, highly optimized, and extended version of Prolog.

B. APPROACH

What is expected from an expert system is to follow the thought process of a typical human expert. As expressed earlier, the design guidelines could be based on either moving from the **first principles** or applying the **experiential approach**. Moving from the first principles which would require cumbersome techniques to simulate the complex nature of engine system, and so is unapplicable. Since both a wide variety of general-purpose car repair/diagnosis books and the past technical experience of the designer in engine maintenance field are available, **the experiential approach** was chosen.

We next evaluated the quality of information available in repair books and converted it into the more-convenient rules and facts format. The design stages were decomposed in the order of thought processes of a typical human expert. The first thing to do was to introduce into the computer the observed behavioral discrepancies in some way. A troubleshooting chart which collected the most common symptoms under two general fault cases was mapped to computer so that when the program is executed, it gives a menu asking for the case, gets symptoms from the user, and caches them.

Once the symptoms are gathered interactively, the diagnostic reasoning process should start. For instance, suppose that a car engine will not start. There are numerous reasons that an engine will not start, from an electrical failure to a faulty fuel system. For that reason, the second step as followed by both a typical human expert and the troubleshooting charts is to identify the subsystems which may explain the behavior and sort them in a reasonable order. This was carried out by defining an embedded table of pairs, each illustrating subsystems for each symptom.

For the third stage of the system-developing phase, following the procedural nature of troubleshooting charts was not appropriate as discussed in Chapter II. It was at this point that a human expert and troubleshooting charts differ from each other. Starting from the most problem-related subsystem, we needed to search a known fault database for that subsystem to find out faults which could account for the same behavioral discrepancies. At this point we needed a **partitioned control structure**. Technical and heuristic information of the engine system was divided into separate files containing rules and facts so that each group would have minimal interactions with other groups. One more partition was needed to hold the global data and general control structures.

Some subsections showed different implementation problems than others. For instance, due to the serializable nature of defects in the battery section, each would show the same symptoms; an **exhaustive search** mechanism was used to test them in a reasonable order. On the other hand, a **rule-cycle-hybrid** control structure was used for the ignition system to test hypotheses.

Since faults within each subsystem were unique to it, fault lists were used, as for instance:

list_of_expected_diagnosis (battery, [case_cracked, case_intact, discharged_battery]).

list_of_expected_diagnosis (battery_cable_connections, [open_circuit, bad_cable_connections]).

Testing a hypothetical or actual defect is the same. The human expert must find a method which will reveal whether it exists or not. For that reason, we have a list of some recommended methods gathered from repair books, for each defect under consideration, as in the following:

recommended ([visual_inspection, high_beam, voltmeter], battery, discharged_battery).

recommended([visual_inspection,voltmeter], battery_cable_connections, open_circuit).

Since we do not assign partial certainly to how precise a method is, our expert system attempts each method in turn.

There are a couple of comments that we would like to emphasize. As could be noticed in the above format, the

same equipment can be used to verify more than one defect, but this is rare. That suggests unique definition and treatment for each operator. And each operator under consideration passes through the three unique steps:

1. **Satisfy Operator Preconditions**: For each operator, a precondition list is formed, such that an operator would be applied only after achieving each member of precondition list. This means:
   - To check if an operator is available.
   - If an operator is available, to check if the user is capable of using this operator.
   - So as to avoid wrong conclusions, to check what other system components should be okay before applying the operator, taking into consideration the way the operator will be applied and the expected measurement.
   - And finally, to give precedence to system components which are preconditions of other components, to avoid unnecessary user dialogue.

   Some example preconditions:

preconditions (visual_inspecton,battery_cable_connections, open_circuit,[ ]).

preconditions (voltmeter,battery_cable_connections, bad_cable_connections, [not (defective (battery)), not(electrical_circuit_problem), not (unable_to_use (voltmeter))]).

2. Procedure Test: Once the preconditions for an operator in use are accomplished, the specific procedure or method is diplayed.

3. Verify Diagnosis: A specific measurement is gathered from the user, a measurement that should be consistent with the data previously displayed.

In testing and pruning the possible defects, the diagnostic reasoning process will come to an end if some are found. Then the program displays verified diagnostic results, ordered by the number of methods used to prove them.

C. KNOWLEDGE BASE

As stated earlier, the knowledge base of our expert system is composed of partitions, each similar in structure. This enables us to maintain, update and debug easily only the necessary partitions.

Procedural knowledge and heuristic knowledge are the two types of information forming the knowledge base. As discussed before, there was an inadequate amount of heuristic information in the repair book in the BATTERY and STARTING sections, so the partitions of those subsystems were only procedural knowledge, whereas the IGNITION system is composed of both types of information.

D. INFERENCE PROCEDURE

Two different inference engines were applied in implementation. The procedure designated as **inference-engine 1** in the GLOBAL file does an exhaustive search to test each defect in currently active knowledge base.

For those partitions which are going to be driven by the exhaustive search mechanism, ordering the faults to be tested is important, for two reasons:

1. It is a good idea to assign high priority to testing defects which are thought to be most likely.
2. It is good to test preconditions of an operator before testing this operator. Otherwise, the expert system will be unfocussed among components to be verified.

The second procedure is designated as **inference-engine 2** in the GLOBAL file. The rule-cycle-hybrid control

structure written, in Prolog, by Rowe [Ref. 5] is variant version of this procedure. Once hypotheses are gathered, inference-engine 2 activates a procedure which will test each hypothesis systematically.

A sample rule written in a format recognizable by the rule-cycle-hybrid (from IGNITION knowledge base) is:

```
hr: - not (hypothesis (incorrect (distributor_firing_
    sequence))),
    fact (hard_starting),
    fact (backfire_through_the_carburetor),
        askif (recent_operator_job),
        asserta (hypothesis (incorrect (distributor_firing
        _sequence))).
```

The rule order for rules governed by the rule-cycle-hybrid is important. A rule is given precedence in database order over the rules whose right sides mention a predicate to be asserted by it.

E. USER INTERFACE

Three modules are required to support our user interface. The first procedure **ask-which** in the GLOBAL file was written originally by Rowe [Ref. 5] but was adapted. The procedure **ask-which** gathers symptoms, behavioral discrepancies observed by user. When it is invoked, it gives a menu, four questions at a time, and asks which questions should be answered yes. After getting the

answers, the invoking procedure checks for contradictory answers, consulting a table of contradictory sets. If there is some contradiction, the user is warned. When all done with input, the symptoms gathered are added to current database by the **asserta** built-in predicate.

The second predicate **askif** of one argument was written originally by Rowe [Ref. 5] but, was adapted again with some minor modifications. This procedure gets virtual facts (facts demanded only when needed). The user is prompted by some question and is expected to answer either affirmatively or negatively. Otherwise it complains and asks for a reasonable answer. Also the answer is cached, so as to not ask the same question again.

The third procedure is code-interpreter. This procedure displays the text of information related to a method to be followed by user to achieve a measurement. If the user is not satisfied with the information supplied, it gives the user further explanation, if available.

F. MODULES

Our expert system is made up of partitioned knowledge bases. While the knowledge bases for BATTERY, STARTER and IGNITION sections are fully implemented, each under the same filename as their section names, the knowledge bases for FUEL and COMPRESSION sections were not implemented due to time constraints. But to show the features and capability

of a vehicle diagnosis expert system, those unimplemented knowledge bases are suggested in the ENGINE partition for one particular fault case.

The partitioned knowledge bases are necessary not only for ease of maintenance and debugging, but also to support different control structures for each. For that reason, rather than using the **consult** built-in predicate, a **reconsult** predicate (which replaces the predicates currently in database with new ones) is always used.

One interesting and powerful feature of our expert system occurs when the rules belonging to current partition need to access momentarily some other partition for specific information about some component. Then the current process state is saved by the **save** built-in predicate, and the information referring to the current process state is pushed onto a process stack, and the new partition is brought in by the **reconsult** predicate. When all done, the original process state is resumed by the **restore** built-in predicate.

IV. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

Arity/Prolog appears suitable for this implementation, and the rich built-in predicates available were helpful. However, the deficiency of error-checking mechanisms against minor misspellings leads to bizarre unintended effects.

The reasoning process in repair manuals which is not suitable for computers was effectively converted into a rule-based model so that the thought process of a typical human expert was efficiently simulated. So a fully implemented diagnostic expert system could be used in place of a human expert.

B. RECOMMENDATIONS

As is clear from this thesis, a full implementation of our system is quite possible. A better diagnostic expert system could support a user by graphic enhancements. The location of a component, the necessary steps to access the component, and some specific procedures to make a measurement could be displayed to an unexperienced mechanic graphically.

An expert system should be able to explain its reasoning path for teaching and debugging purposes. We could enhance our expert system with that feature.

APPENDIX A

SOURCE CODE

This appendix contains a listing of the main program (held in the GLOBAL, BATTERY, STARTER, IGNITION, and ENGINE section files which contain the knowledge base of the expert system).

This expert system implementation was written in the version of the PROLOG language known as ARITY-PROLOG (which is a product of Arity Corporation) and runs under the MS-DOS operating system on IBM PC/XT clones. This version of PROLOG is closely based on standards as described in Clocksin and Mellish [Ref. 11].

Having entered the PROLOG, the program comes up with a short message about start-up and then the user starts the consultation with the query of "diagnosis." The lines that are limited with "/*" are comment lines. They should not be confused with actual PROLOG code.

%THE FOLLOWING CONTAINS THE CONTENTS OF GLOBAL FILE.

```
/****MAIN PROGRAM****

:- cls,nl,nl,nl,
   write($  VEHICLE DIAGNOSIS EXPERT SYSTEM IMPLEMENTATION   &),
   nl,
   write($ Please enter "diagnosis" to start the consultation. &),
   nl,nl,nl.

/*
   The predicate "DIAGNOSIS" is the top level predicate to start the expert system.
/*

diagnosis :- procedure_list (LIST), run_diagnosis(LIST),
             findall (X,proved(diagnosis(X,Y)),L),
             diagnosis_likelihood_list(L,LM,LL),
             write($ MOST LIKELY DIAGNOSIS LIST : $),nl,nl,
             write_list(LM),write($ LESS LIKELY DIAGNOSIS LIST :$),
             nl,nl,write_list(LL).

/*
   Having been supplied by the symptoms, by means of early dialogue with consultor,
   the problem state is identified and relevant subsections of the engine system of the
   vehicle to be considered are held within a list, which is the argument of predicate
   "PROCEDURE-LIST" in a reasonable order.  At this point, the general expert
   approach is to search those sections one by one in the order they were thought to be
   reasonable, until some specific diagnostics are found.
/*
run_diagnosis(LIST) :- proved(diagnosis(Diagnosis,Equipment)).
run_diagnosis([]).
run_diangosis([Section|Sections])  :-
                          file_table(Section,Section_File),
                          write($***  $),write(Section_File),
                          write($  SECTION  ***$),nl,nl
                          reconsult(Section_File),
                          restore_database(Section_File)
                          section_engine_table(Section,Inference_Engine),
                          call_engine(Inference_Engine),
                          run_diagnosis(Sections).
run_diagnosis([Section:Sections])  :-
                          not(file_table(Section,Section_File)),
                          write($***  $),write(Section),
                          write(  has not been implemented yet.   ***$),
                          nl,run_diagnosis(Sections).

call_engine(Inference_Engine)  :- call(Inference_Engine).
call_engine(Inference_Engine).

/*
   The predicate "PROCEDURE-LIST" initiates the communication with consultor,
   using menu-driven interaction method, at the very early program stage, to obtain
   the symptoms to create existing facts and assess subsections of engine system to be
   considered related to problem state.
/*
```

```
procedure_list(LIST)  :- start_up_menu(Answer_List),
                         create_facts(Answer_List),
                         list_finder(Answer_List,L),sort(L,L1),
                         !,list_converter(L1,LIST).


start_up_menu(Answer_List)  :- repeat,
                               [!ask_which([engine_wont_start,
                               engine_runs_poorly],List!],
                               not(contradiction(List)),
                               create_facts(List),!,
                               sub_menu(List,Answer_List).

sub_menu(List,Answer_List)  :- member(engine_wont_start,List),repeat,
                              [!ask_which([starter_doesnt_turn_at_all,
                              starter_turns_but_not_the_engine,
                              slow_cranking,normal_cranking,quick_cranking,
                              engine_fires_intermittently,
                              engine_fires_consistently],Answer_List)!],
                              not(contradiction(Answer_List)).
sub_menu(List,Answer_List)  :-member(engine_runs_poorly,List),repeat,[!
ask_which([hard_starting,rough_idle,stalling,
                              engine_dies_at_high_speed,
                              hesitation_on_acceleration,
                              poor_pick_up,lack_of_power,
                              backfire_through_the_carburator,
                              backfire_through_the_exhaust,
                              blue_exhaust_gases,black_exhause_gases,
                              running_on,susceptible_to_moisture,
                              misfire_under_load,engine_miss_at_high_rpm,
                              misfire_at_idle_speed],
Answer_List)!],not(contradiction(Answer_List)).


ask_which([A,B,C,D|L],List)  :-
                        screen_ask_which([A,B,C,D],[A,B,C,D],Lst),
                        ask_which(L,Intlist),append(Lst,Intlist,List).
ask_which([],[]).
ask_which(L,List)  :- length(L,N),N<4,N>0,
                           screen_ask_which(L,L,List).

screen_ask_which([X|L),L2,Lst)  :- length(L,N),length(L2,N2),
        N3 is N2-N,write(N3),write(':'),questioncode(X,Q),write(Q),
                                          write('?'),nl,screen_ask_which(L,L2,Lst).
screen_ask_which([],L2,Lst)  :-
        write($ Give numbers of questions whose answer is yes.$),
        read(AL),nl,create_list(AL,L2,Lst).

create_list([],L2,[]).
create_list([N|L],L2,[I|L3])  :- item(N,L2,I),
                                       create_list(L,L2,L3).
create_list([N|L],L2,L3)  :- not(item(N,L2,I)),
                                    create_list(L,L2,L3).
```

```
item(1,[X|L],X).
item(N,[X|L],I)  :- N>1,N2 is N-1,item (N2,L,I).


contradiction(List)  :- length(List,N),N>1,
                              contradictory_list(X,L),
                              [!member(X,List)!],not(list_done([X],List,L)),
                              message.

list_done(A,B,[]).
list_done(A,B,[C|D])  :- union(A,[C],E),!,not(subset(E,B)),

                              list_done(A,B,D).



create_facts([])  :- !.
create_facts(Fact|Answer_List])  :-
                        asserta(fact(Fact)),crate_facts(Answer_List).


list_finder([X|L1],L)  :- check_list(X,List),list_finder(L1,L2),
                              union(List,L2,L).
list_finder([],[]).

list_converter([],[]).
list_converter([X|L],[Y|L1])  :- convert(X,Y),
                                    list_converter(L,L1).

/*
   The complicated nature and peculiarities of subsections of the engine system
   necessitate more than one inference-engine to process the defined subsections.
   INFERENCE-ENGINE1 simulates the role of "exhaustive-search-mechanism" for
   the section it was allowed to process.
/*


inference_engine1  :- not(all_done).

all_done :-
            list_of_expected_diagnosis(Part,Expected_Diagnosis_List),
            attempt_to_verify_all(Part,Expected_Diagnosis_List),fail.

attempt_to_verify_all(Part,[]).
attempt_to_verify_all(Part,
                        [First_Diagnosis|Expected_Diagnosis_List])  :-
                        verify_diagnosis(Part,First_Diagnosis),
                      attempt_to_verify_all(Part,Expected_Diagnosis_List).

verify_diagnosis(Part,First_Diagnosis)  :-
                    recommended(Equipment_List,Part,First_Diagnosis),
        verify_using_all(Equipment_List,Part,First_Diagnosis).
verify_using_all([],Part,First_Diagnosis).
verify_using_all([First_Equipment:Equipment_List],Part,
```

```
                                                    First_Diagnosis) :-
        apply_equipment(First_Equipment,Part,First_Diagnosis),
        verify_using_all(Equipment_List,Part,First_Diagnosis).


apply_equipment(First_Equipment,Part,First_Diagnosis) :-
   preconditions(First-Equipment,Part,First_Diagnosis,
                                                        Precondition_List),
   satisfied_preconditions(Precondition_List),find(Part),
   get_ready_equipment(First_Equipment,Part,First_Diagnosis),
   prove_diagnosis(First_Equipment,Part,First_Diagnosis).

apply_equipment(First_Equipment,Part,First_Diagnosis) :-
preconditions(First_Equipment,Part,First_Diagnosis,
                                                  Precondition_List),
       not(satisfied_preconditions(Precondition_List)).
satisfied_preconditions([]).
satisfied_preconditions([First_Precondition:Precondition_List])
   :- call_precondition(First_Precondition),
satisfied_preconditions(Precondition_List).
prove_diagnosis(First_Equipment,Part,First_Diagnosis) :-
       proved_diagnosis(First_Equipment,Part,First_Diagnosis),
       update_diagnosis_database(First_Equipment,First_Diagnosis),
prove_diagnosis(First_Equipment,Part,First_Diagnosis) :-
       not(proved_diagnosis(First_Equipment,Part,First_Diagnosis)).
update_diagnosis_database(First_Equipment,First_Diagnosis) :-
      not(proved(diagnosis(First_Diagnosis,First_Equipment))),
      asserta(proved(diagnosis(First_Diagnosis,First_Equipment))).
update_diagnosis_database(First_Equipment,First_Diagnosis) :-
      proved(diagnosis(First_Diagnosis,First_Equipment)).

/*
   The predicate "DIAGNOSIS" of one argument searches for the malfunction in
   particular, provided that the name of the malfunction is supplied as its argument.
/*

diagnosis(Diagnosis) :- proved(diagnosis(Diagnosis,X)).
diagnosis(Daignosis) :- list_of_expected_diagnosis(Part,
                          Expected_Diagnosis_List),
                        member(Diagnosis,Expected_Diagnosis_List),
                        verify_diagnosis(Part,Diagnosis),!,
                        proved(diagnosis(Diagnosis,X)).

/*
   INFERENCE-ENGINE2 simulates the role of "rule-cycle-hybrid control
   mechanism" for the section it was allowed to process.
/*

inference_engine2 :- hybrid,!,doall_until.

done :- hypothesis(Diagnosis).
hybrid :- done.
hybrid :- not(one_cycle),flag,abolish(flag/0),hybrid.
one_cycle :-hr,asserta(flag),fail.
```

```
doall_until :- doall,not(hypothesis(Diagnosis)).

doall :- not(alltried).
alltried :-call(test_hypothesis),fail.


test_hypothesis :-hypothesis(Diagnosis),
                        [!retract(hypothesis(Diagnosis))!],
                        diagnosis(Diagnosis).

/*
   The predicate "DIAGNOSIS-LIKELIHOOD-LIST" sorts the malfunctions whose
   existence has been proved by the expert system, and assigns them likelihoods
   according to the number of methods used to locate them.
/*


diagnosis_likelihood-list([],[],[]).
diagnosis_likelihood_list([X],[],[X]).
diagnosis_likelihood_list([X|L],[X|L2],L3) :-
                                member(X,L),delete(X,L,L1),
                                diagnosis_likelihood_list(L1,L2,L3).
diagnosis_likelihood_list([X|L],L1,[X|L2]) :-
                                not(member(X,L)),
                                diagnosis_likelihood_list(L,L1,L2).

/*
   The predicate "ASKIF" of one input argument performs the interaction between
   expert system and consultor.  It will have only one argument, the question, and it
   will succeed if that question is answered affirmatively and fail if the question is
   answered negatively.  If an answer is unclear, it will complain and ask for another
   answer.
/*

askif(A) :-
        ask(A,B),
        positive_answer(B).

askifnot(A) :-
        not askif(A).

ask(A,B) :-
        asked(A,B).
ask(A,B) :-
        not asked(A,B),
        (questioncode(A,C);global_questioncode(A,C),
        write(C),
        write($?            (yes/no) ==>  $),
        read(D),
        nl,
        [! answer(D.B)
        !],
        not unexpected_answer(B),
        asserta(asked(A,B)).
```

```
ask(A,B) :-
      not asked(A,B),
      not questioncode(A,C),not global_questioncode(A,C),
      write(A),
      write($?            (yes/no) ==>  $),
      read(D),
      nl,
      [! answer(D,B)
      !],
      not unexpected_answer(B),
      asserta(asked(A,B)).


answer(A,B) :-
      not unexpected_answer(A),
      B=A

answer(A,B) :-
      not unexpected_answer(A),
      repeat,
      write($ Please answer yes or no ==>  $),
      read(B),
      nl,
      not unexpected_answer(B).

unexpected_answer(A) :-
      not affirmative(A),
      not negative(A),

positive_answer(A) :-
      affirmative(A).

/*
   The predicate "CODE-INTERPRETER" provides the consultor with the domain-
   dependent information.
/*

code_interpreter(A) :-
        expressed(A).
code_interpreter(A) :-
        not expressed(A),
        code(A,B,C),
        write(B),
        nl,
        asserta(expressed(A)),
        further_explain(C).

further_explain($No more$) :- !.
further_explain (A) :-
      askif(need_explain),
      retract(asked(need_explain,B)),
      write(A),
      nl.
further_explain(A) :-
      askifnot(need_explain),
```

```
        retract(asked(need_explain,B)).


delete(X,[],[]).
delete(X,[X|L],M) :- delete(X,L,M).
delete(X,[Y|L],[Y|M]) :- delete(X,L,M).

append([],L,L).
append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).


subset([],A).
subset([A|B],C) :- member(A,C),subset(B,C).

member(X,[X|L]).
member(X,[Y|L]) :- member(X,L).

union([],X,X).
union([X|R],Y,Z) :- member (X,Y),!,union(R,Y,Z).
union ([X|R],Y,[X|Z]) :- union(R,Y,Z).

write_list([]) :- !.
write_list([X|Y]) :- write (X),nl,write_list(Y).

first([X|L],X).

second([X,Y|L],Y).

/*
   The fact "CHECK-LIST" holds the list of subsections of engine system, determined
   hypothetically, concerning each specific symptom.
/*

check_list(starter_doesnt_turn_at_all,[10,20]).
check_list(starter_turns_but_not-the_engine,[20]).
check_list(slow_cranking,[10,20]).
check_list(normal_cranking,[30]).
check_list(quick_cranking,[50]).
check_list(engine_fires_intermittently,[30]).
check_list(engine_fires_consistently,[40,50]).
check_list(hard_starting,[30,40]).
check_list(rough_idle,[30,40]).
check_list(stalling,[30,40]).
check_list(engine_dies_at_high_speed,[30,40]).
check_list(hesitation_on_acceleration,[30.40]).
check_list(poor_pick_up,[30,40]).
check_lsit(lack_of_power,[30,40]).
check_list(backfire_through_the_carburator,[30,70]).
check_list(backfire_through_the_exhaust,[30,70]).
check_list(blue-exhaust_gases,[50,60]).
check_list(black-exhaust_gases,[40]).
check_list(running_on,[30]).
check_list(susceptible_to_moisture,[30]).
check_list(misfire_under_load,[30,60,70]).
```

```
check_list(engine_miss_at_high_rpm,[30]).
check_list(misfire_at_idle_speed,[30,40.60]).

/*
   The fact "CONTRADICTORY-LIST" holds the list of symptoms for each symptom, to
   establish that a symptom in question is a member of the set of possible symptoms.
/*

contradictory_list(engine_wont_start,[engine_runs_poorly]).
contradictory_list(starter_doesnt_turn_at_all,
                   [starter_turns_but_not-the_engine,slow_cranking,
                   normal_cranking,quick_cranking,
                   engine_fires_intermittently,
                   engine_fires_consistently]).
contradictory_list(starter_turns_but_not_the_engine,
                   [slow_cranking,normal_cranking,quick_cranking,
                   engnie_fires_intermittently,
                   engine_fires_consisstently]).
contradictory_list(slow_cranking,[normal_cranking,quick_cranking]).
contradictory_list(normal_cranking,[quick_cranking]).

contradictory_list(engine_fires_intermittently,
                   [engine_fires_consistently]).
contradictory_list(blue_exhaust_gases,[black_exhaust_gases]).

convert(10,battery).
convert(20,starting_system).
convert(30,ignition_system).
convert(40,fuel system).
convert(50,engine_compression).
convert(60,engine_vacuum).
convert(70,valve-train).



find(sub_diagnosis).
find(complete_system).
find(ignition_switch).
find(Part) :- askif(where(Part),!.
find(Part) :- code_interpreter(Part),retract(asked(where(Part),

                              X)),assert(asked(where(Part),yes)).

unable_to_use(X) :- askifnot(has(X)).
unable_to_use(X) :- askifnot(how_to_use(X,known)).


affirmative(yes).
affirmativey(y).
affirmative(right).
affirmative(okay).

negative(no).
negative(n).
negative(not).
```

```
negative(never).
negative(impossible).


global_questioncode(need_explain,
                                    $Do you need further explanation$).
global_questioncode(hear(X),X) :-
                                  write($Did you hear a sound like a $).
global_questioncode(has(X),X) :-
                                                  write($ Do you have a $).
global_questioncode(how_to_use(X,Y),X) :-
                                    write($ Do you know how to use a $).
global_questioncode(where(Part),$$) :-
     write($Do you know where the $),write(Part),write($ is$).
questioncode(engine_wont_start,$ Does the engine start at all$).
questioncode(engine_runs_poorly,$ Does the engine run poorly$).
questioncode(starter_doesnt_turn_at_all,
                                  $ Does starter turn at all$).
questioncode(starter_turns_but_not_the_engine,
                        $ Does starter turn, but not the engine$).

questioncode(slow_cranking,
                          $ Does starter turn engine very slowly$).
questioncode(normal_cranking,$ Does starter turn engine normally$).
questioncode(quick_cranking,$ Does starter turn engine very quickly$).
questioncode(engine_fires_intermittently,
                                    $ Does engine fire intermittently$).
questioncode(engine_fires_consistently,
                                      $ Does engine fire consistently$).
questioncode(hard_starting,$ Do you have hard starting problem$).
questioncode(rough_idle,$ Do you have a rough idle$).
questiocode(stalling,$ Do you have stalling$).
questioncode(engine_dies_at_high_speed,
                                      $ Does engine die at high speed$).
questioncode(hesitation_on_acceleration,
                           $ Do you have hesitation (on accelaration
                             from standing stop)$).
questioncode(poor_pick_up,$ Do you have poor pickup$).
questioncode(lack_of_power,$ Do you have lack of power$).
questioncode(backfire_through_the_carburator,
                      $ Do you have backfire through the carburator$).
questioncode(backfire_through_the  exhaust,
                         $ Do you have backfire through the exhaust$).
questioncode(blue_exhaust_gases,$ Do you have blue exhaust gases$).
questioncode(black_exhause_gases,$ Do you have black exhaust gases$).
questioncode(running_on,$ Do you have running on ( after the ignition
                                                  is shut off)$).
questioncode(susceptible_to_moisture,$ Is it susceptible to moisture$).
questincode(misfire_under_load,$ Does the engine misfire under load$).
questioncode(engine_miss_at_high_rpm,
                                    $ Does the engine misfire at speed$).
questioncode(misfire_at_idle_speed,
                                 $ Does the engine misfire at idle speed$).
```

```
message :- write($I found a contradiction!$),nl,
     write($ Check and repeat your answer.$),
     nl.

/*
   Since rule and fact partitioning were chosen to be design guidelines for the
   knowledge base of the expert system, the table "FILE-TABLE" holds the file name of
   each partition to be accessed by "RECONSULT" system predicate of one argument.
/*

file_table(battery,battery).
file_table(starting_system,starter).
file_table(ignition_system,ignition).
file_table(engine_compression,engine).

/*
   The table "SECTION-ENGINE-TABLE" holds the name of specific inference-engine
   to process the partition in question.
/*

section_engine_table(battery,inference_engine1).
section_engine_table(starting_system,inference_engine1).
section_engine_table(ignition_system,inference_engine2).
section_engine_table(engine_compression,inference_engine1).

/*
   "PREDICATE-FILE-INTERFACE" designates the location of intermediate-predicate
   to be interfaced among the partitions.
/*

predicate_file_interface(defective(battery),battery).
predicate_file_interface(electrical_circuit_problem,battery).
predicate_file_interface(high_resistance_in_engine,engine).

/*
   In order to be able to access temporarily the partition holding the intermediate-
   predicate neatly for some specific information, when the intermediate-predicate to
   be queried is not residing in current process state, the current process state has to be
   pushed onto a process stack and the holding partition should be brought. When all
   done, we should come back to the original process state by popping off the stack,
   carrying all cached facts and conclusions.

   We should also ensure mutual-exclusion condition, since the same procedure
   possessing that logic could be consulted several times at different process states,
   possibly attempting to modify the state information which is supposed to be global
   throughout the program, inevitably. The top-level predicate "CALL-
   PRECONDITION" of one argument which stands for the intermediate-predicate itself
   carries that logic out successfully even for some piling up process states.
/*

call_precondition(not(F_P)) :- clause(F_P),true),!,fail.
call_precondition(not(F_P)) :- call_precondition2(F_P),!,fail.
call_precondition(not(F_P)) :- call_precondition3(F_P),!.
call-precondition(not(F_P)) :- !,not(call(F_P)),!.
call-precondition(F_P) :- clause(F_P,true),!.
```

```
call-precondition(F_P) :- call_precondition2(F_P),!.
call_precondition(F_P) :- call_precondition3(F_P),!,fail.
call-precondition(F_P) :- !,call(F_P),!.


call_precondition2(First_Precondition) :-
     asserta(ticket),
   not(clause(fact(First_Precondition,JJJ),true)),
   predicate_file_interface(First_Precondition,Section_File),
   process_stack(LIST),[!not(first(LIST,[Section_File,X,Y])),
   retract(ticket),first(LIST,[File_Name,XX,File_Attribute]),
   find_world_name(File_Name,File_Attribute,World_Name),
   save(World_Name),world_snap_shot(SN1),
   asserta(cache(World_Name,SN1)),asserta(ticket),
   write($*** LOADING $),
   write(Section_File),write($ SECTION ***$),nl,nl,reconsult(Section_File),
   retract(waiting_process_number(N1)),N2 is N1 + 1,
   asserta(waiting_process_number(N2)),
   retract(process_stack(LIST),append([[Section_File,fail,N2]],
   LIST,FINAL_LIST),asserta(process_stack(FINAL_LIST))!],
   call(First_Precondition),world_snap_shot(SN2),
   write($*** EXITING $),write(Section_File),
   write($ SECTION ***$),nl,nl,create(TEMP,'temp.ari'),
   advance_world(TEMP,SN1,SN2),
   restore(World_Name),[temp],delete('temp.ari'),
   asserta(fact(First_Precondition,yes)),
   !.



call_precondition3(First_Precondition) :-
   [!retract(ticket)!],not(ticket),
   not(clause(fact(First_Precondition,JJJ),true)),
   predicate_file_interface(First_Precondition,Section_File),
   waiting_process_number(N1),
   Process_stack(LIST),first(LIST,[Section_File,fail,N1]),
   write($*** UNLOADING $),write(Section_File),
   write($ SECTION ***$),nl,nl,
   second(LIST,[File_Name,XXX,File-Attribute]),
   find_world_name(File_Name,File_Attribute,
   World_Name),retract(cache(World_Name,SN1)),
   world_snap_shot(SN2),create(TEMP,'temp.ari'),
   advance_world(TEMP,SN1,SN2),
   restore(World_Name),[temp],
   asserta(fact(First_Precondition,no)),delete('temp.ari'),!.

/*
   The predicate "ADVANCE-WORLD" serves to advance the database of the previous
   state with the facts asserted and conclusions reached, while still keeping the
   database consistent.
/*

advance_world(TEMP,LIST1,LIST2) :-
                                    write_cache_list(TEMP,LIST1,LIST2),!.
```

```
write_cache_list(TEMP,LIST1,LIST2) :- member (X,LIST2),
                    not(member(X,LIST1)),write(TEMP,X),
                    write(TEMP,,),nl(TEMP),fail.
write_cache_list(TEMP,LIST1,LIST2) :- nl(TEMP),close(TEMP).

/*
   The predicate "RESTORE-DATABASE" of one argument accepts the current partition
   name residing as its argument and initializes process-state parameters for that
   partition.
/*

restore_database(Section_File :-
                    abolish(process_stack/1),
                    abolish(waiting_process_number/1),
                    asserta(process_stack([[Section_file,true,0]])),
                    asserta(waiting_process_number(0)).

/*
   The predicate "CACHE-PREDICATE-LIST" keeps  the list of facts, to be carried
   among the process states, once they were cache advanced.
/*

cache_predicate_list([asked(A,B),proved(diagnosis(C,D)),
                          expressed(E),cracked_battery,
                          electrical_circuit_problem,light_problem,
                          defective(battery),
                          rest_of_okay(starting_system),
                          high_resistance_in_engine]).

/*
   The purpose of the predicate "FIND-WORLD-NAME" is to code and decode either
   current or previous process state to a unique file name to be saved or restored.
/*

find_world_name(File_Name,File_Attribute,World_Name)  :-
                    atom_string(File_Name,String1),
                    substring(String1,0,3,String2),
                    int_text(File_Attribute,String3),
                    concat(String2,String3,String4),
                    atom_string(World_Name,String4).

/*
   The predicate "WORLD-SNAP-SHOT" gets the list of whole asserted facts within the
   process state for which it was called.
/*

world_snap_shot(LIST) :- cache_predicate_list(L1),
                           run_snap_shot(L1,LIST),
run_snap_shot([X],LIST) :- local_snap_shot([],X,LIST).

run_snap_shot([X|LIST],LIST2) :-
                                  local_snap_shot([],X,LIST1),
                                  run_snap-shot(LIST,LIST3),
```

```
                                append(LIST1,LIST3,LIST2).

local_snap_shot(A,B,C) :-
                        asserta(sublist(A)),
                        clause(B,true),[!sublist(E),append([B],E,F),
                        retract(sublist(E)),asserta(sublist(F))!],fail.
local_snap_shot(A,B,C) :- sublist(D),retract(sublist(D)),C=D,!.
```

%THE FOLLOWING CONTAINS THE CONTENTS OF BATTERY FILE.

```
list_of_expected_diagnosis(battery,[case_cracked,case_intact,
                                                    discharged_battery]).
list_of_expected_diagnosis(battery_cable_connections,
     [bad_cable_connections,open_circuit]).

recommended([visual_inspection],battery,case_cracked).
recommended([visual_inspection],battery,case_intact).
recommended([high_beam,hydrometer],battery,
                                                    discharged_battery).
recommended([visual_inspection,high_beam,voltmeter],
     battery_cable_connections,bad_cable_connections).
recommended([visual_inspection,voltmeter],
              battery_cable_connections,open_circuit).

preconditions(visual_inspection,battery,case_cracked,[]).
preconditions(visual_inspection,battery,case_intact,[]).
preconditions(high_beam,battery,discharged_battery,
                  [not(cracked_battery),not(light_problem)]).
preconditions(hydrometer,battery,discharged_battery,
                  [not(unable_to_use(hydrometer))]).
predonditions(visual_inspection,battery_cable_connections,
                                       bad_cable_connections,[]).
preconditions(high_beam,battery_cable_connections,
     bad_cable-connections,[not(cracked_battery),
               not(light_problem),not(electrical_circuit_problem)]).
preconditions(voltmeter,battery_cable_connections,
     bad_cable_connections,[not(defective(battery)),
                                       not(electical_circuit_problem),
                                       not(unable_to_use(volmeter))]).
preconditions(visual_inspection,battery_cable_connections,
                                       open_circuit,[]).
preconditions(voltmeter,battery_cable_connections,open_circuit,
                                            [not(defective(battery)),
                                       not(unable_to_use(voltmeter))]).

cracked_battery :- diagnosis(case_cracked),asserta(cracked_battery).
electrical_curcuit_problem :- diagnosis(open_circuit),asserta
                                            (electrical_circuit_problem).
light_problem :- askifnot(light_on),asserta(light_problem).
defective(battery) :- cracked_battery,asserta(defective(battery)).
defective(battery) :- diagnosis(discharged_battery),asserta(defective(battery)).

questioncode(light_on,
$Did the light come out when the knob turned on$).
questioncode(dim,$Did the lights dim considerably or go out$).
questioncode(low_charge,$Does it indicate less than 1.140 @$).
questioncode(cracked,$Is the battery case cracked$).
questioncode(exceeded,$Did voltage drop exceed 0.4 volts$).
questioncode(intact,$Is the battery case intact$).
questioncode(bad,$Did you notice bad cable or connections$).
questioncode(open_circuit,$Did you notice an open connection$).
questioncode(same,$Is the reading same as the battery reading$).
```

```
code(10,$Turn lights on high,
     try starter and note action of lights.$,$No more$).
code(20,$Test the state of charge of
     battery using the hydrometer.$,$No more$).
code(30,$Inspect the battery case. $,
     $For cracks, corrosion and water level.$).
code(40,$Connect prods of voltmeter on 3-volt scale to
     grounded battery post and starter moter housing.
     Close the starter switch and not the voltmeter
     reading.$,$No more$).
code(50,$Inspect the battery cables.$,
     $For loose, broken open  cables and connections:$).
code(battery,$Battery located under the engine hood,
               most probably on right front of the vehicle,$,
               $No more$).
code(battery cable connections,
$Located between battery and starter unit.$,$No more$).

get_ready_equipment(visual_inspection,Part,case_cracked) :-
                                          code_interpreter(30).
get_ready_equipment(visual_inspection,Part,case-intact) :-
                                          code_interpreter(30).
get_ready_equipment(high_beam,Part,discharged_battery) :-
                                          code_interpreter(10).
get_ready_equipment(hydrometer,Part,discharged_battery) :-
                                          code_interpreter(20).
get_ready_equipment(visual_inspection,Part,bad_cable_connections)
                                          :- code_interpreter(50).
get_ready_equipment(high_beam,Part,bad_cable_connections) :-
                                           code_interpreter(10).
get_ready_equipment(voltmeter,Part,bad_cable_connections) :-
                                           code_interpreter(40).
get_ready_equipment(visual_inspection,P,open_circuit) :-
                                           code_interpreter(50).
get_ready_equipment(voltmeter,Part,open_circuit) :-
                                           code_interpreter(40).


proved_diagnosis(visual_inspection,Part,case_cracked) :-
                                           askif(cracked).
proved_diagnosis(visual_inspection,Part,case_intact) :-
                                           askif(intact).
proved_diagnosis(high_beam,Part,discharged_battery) :-
                              askif(dim),askif(hear(clattering)).
proved_diagnøsis(hydrometer,Part,discharged_battery) :-
                                           askif(low_charge).

proved_diagnosis(visual_inspection,Part,bad_cable_connections) :-
                                           askif(bad).
proved_diagnosis(high_beam,Part,bad_cable_connections) :-
                                           askif(dim).
proved_diagnosis(voltmeter,Part,bad_cable_connections) :-
                                           askif(exceeded).
proved_diagnosis(visual_inspection,Part,open_circuit) :-
```

```
                                        askif(open circuit).
proved_diagnosis(voltmeter,Part,open_circuit) :-
                                        askif(same).
```

%THE FOLLOWING CONTAINS THE CONTENTS OF STARTER FILE.

```
list_of_expected_diagnosis(ignition_switch,
                [broken_ignition_switch_connections]).
list_of_expected_diagnosis(magnetic_switch,
                [improperly_functioning_magnetic_switch]).
list_of_expected_diagnosis(starter_solenoid,
                [malfunction_in_starter_solenoid]).
list_of_expected_diagnosis(starter_unit,
                [malfunction_in_starter_unit]).

recommended([electrical_test1,electrical_test2],ignition_switch,
                        broken_ignition_switch_connections).
recommended([electrical_test],magnetic_switch,
                        improperly_functioning_magnetic_switch).
recommended([electrical_test],starter_solenoid,
                         malfunction_in_starter_solenoid).
recommended([symptom(slow_cranking),symptom(starter_spins_free),
symptom(early),deductive_reasoning],starter_unit,
                                        malfunction_in_starter_unit).

preconditions(electrical_test1,ignition_switch,
     broken_ignition_switch_connections,
     [fact(starter_doesnt_turn_at_all),not(defective(battery)),
     (askif(has(test_lamp_12V));not(unable_to_use(voltmeter)))]).
preconditions(electrical_test2,ignition_switch,
     broken_ignition_swith_connections,
     [fact(starter_doesnt_turn_at_all),not(defective(battery)),
     (askif(has(test_lamp12V));not(unable_to_use(voltmeter)))]).
preconditions(electrical_test,magnetic_switch,
     improperly_functioning_magnetic _switch,
     [fact(starter_doesnt_turn_at_all),
     not(diagnosis(broken_ignition_switch_connctions)),
     askif(has(jumper))]).
preconditions(electrical_test,starter_solenoid,
                    malfunction_in_starter_solenoid,
          [fact(starter_doesnt_turn_at_all),not(defective(battery)),
          not(electrical circuit problem),askif(has(jumper))]).
preconditions(symptom(slow-cranking),starter-unit,
          malfunction_in_starter_unit.
          [fact(slow_cranking,not(defective(battery)).
          not(high_resistance_in_engine)]).
preconditions(symptom(starter_spins_free),starter_unit,
          malfunction_in_starter_unit.
     [fact(starter_turns_but_not_the_engine),askif(starter_spins_free)]).
preconditions(symptom(early),starter_unit,
   [fact(starter_turns_but_not_the_engine),not(high_resistance_in_engine)]).


preconditions(deductive_reasoning,starter_unit,
                     malfunction_in_starter_unit,
                     [fact(starter_doesnt_turn_at_all),not(defective(battery)),
     not(electrical_circuit_problem),
     rest_of_okay(starting_system)]).
```

```
rest_of_okay(starting_system) :-
     not(diagnosis(broken_ignition_switch_connections)).
     not(diagnosis(improperly_functioning_magnetic_switch)),
     not(diagnosis(malfunction_in_starter_solenoid)),
                                        asserta(rest_of_okay(starting_system)).


questioncode(lamp_lights,$Did the lamp light or meter needle
                    move, when the switch is turned$).
questioncode(lamp_flickers,$Did the lamp flicker or meter needle
                    move,when the key jiggled$).
questioncode(starter_spins_free,$Is that your problem;
                    starter spins free but won't engage$).
questioncode(starter_operates,$Did the starter operate$).
questioncode(response_recorded,$Did the starter turn the engine
                                        either normally or slowly or buzz$).
code(magnetic_switch,$An electrically operated switch whose only function
                    is to make contact for the starter.May be located on
                    the starter,on the engine side of firewall, or on the
                    fender apron.$,$No more$).
code(starter_solenoid,$An electrically operated whose function is to make
                    electrical contact the the starter,and in addition
                         shift the starter clutch into mesh with the flywheel.
                         Always located on the starter.$,$No more$).
code(starter_unit,$Always located either on right or left side of engine very
                   close to the bottom.$,No more$).
code(100,$Check the ignition switch for loose connections,
     cracked insulation, or broken wires.$,
     $Connect a 12V test lamp or voltmeter between the
     starter post of solenoid and ground.  Turn the ignition
     switch to the "start" position and jiggle the key.$).
code(200,$Determine whether the magnetic switch is functioning
     properly.$,
     $By connecting a jumper across the switch and
     turning the ignition switch to start.$).
code(300,$Test the starter solenoid.$,
     $Connect a jumper from the battery post of
     solenoid to the starter post of solenoid.$).

get_ready_equipment(Test,ignition_switch,
   broken_ignition_switch_connections) :- code_interpreter(100).
get_ready_equipment(electrical_test,magnetic_switch,
improperly_functioning_magnetic_switch) :- code_interpreter(200).
get_ready_equipment(electrical_test,starter_solenoid,
  malfunction_in_starter_solenoid) :- code interpreter(300).
get_ready_equipment(Symptom,starter_unit,
                                        malfunction_in_starter_unit).

proved_diagnosis(electrical_test1,ignition_switch,
       broken_ignition_switch_connections) :- askifnot(lamp_lights).
proved_diagnosis(electrical_test2,ignition_switch,
       broken_ignition_switch_connections) :- askif(lamp_flickers).
proved_diagnosis(electrical_test,magnetic_switch,
```

```
        improperly_functioning_magnetic_switch)  :-
                                                    askif(starter_operates).
proved_diagnosis(electrical_test,starter_solenoid,
                                malfunction_in_starter_solenoid)  :-
                                                    askifnot(response_recorded).
proved_diagnosis(System,starter_unit,
                                                malfunction_in_starter_unit).
```

%THE FOLLOWING CONTAINS THE CONTENTS OF IGNITION FILE.

```
list_of_expected_diagnosis(sub_diagnosis,
[defective(ignition_system),defective(primary_circuit_coil_side),
                    defective(primary_circuit_distributor_side)]).
list_of_expected_diagnosis(ignition_points,
                                    [burned_or_damaged(ignition_points),
                                    out_of_adjustment(ignition_points)]).
list_of_expected_diagnosis(condenser,[defective(condenser)]).
list_of_expected_diagnosis(ballast,_resistor,
                                              [defective(ballast_resistor)]).
list_of_expected_diagnosis(ignition_switch,
                                              [defective(ignition_switch)]).
list_of_expected_diagnosis(ignition_coil,
                                    [defective(coil_primary_resistance),
                                  defective(coil secondary_resistance)]).
list_of_expected_diagnosis(plugs,[defective(spark plugs)])
list_of_expected_diagnosis(distributor,[poor (distributor_ground),
      defective(distributor_rotor),moisture_on(distributor_cap),
                              cracked_or_tracked(distributor_cap),
            defective(distributor_wires_or_ignition_plugs)]).
list_of_expected_diagnosis(complete system,
                                              [induction_firing_of_cyclinders
            incorrect(distributor_firing_sequence),
                        incorrect(ignition_timing)]).

recommended([spark_test],sub_diagnosis,defective(ignition_system)).
recommended([voltmeter_test1,voltmeter_test2],sub_diagnosis,
                                    defective(primary_circuit_coil_side)).
recommended([voltmeter],sub_diagnosis,
                          defective(primary_circuit_distributor_side)).
recommended([visual_inspection],ignition_points,
                                     burned_or_damaged(ignition_points)).
recommended([visual_inspection,dwell_meter],ignition_points,
                                     out_of_adjustment(ignition_points)),
recommended([voltmeter],ballast_resistor,
                                              defective(ballast_resistor)).
recommended([voltmeter},condenser,defective(condenser)).
recommended([electrical_test],ignition_switch,
                                       defective(ignition_switch)).
recommended([voltmeter],ignition_coil,
                                    defective(coil_primary_resistance)).
recommended([voltmeter],ignition_coil,
                                    defective(coil_secondary_resistance)).
recommended([visual_inspection],plugs,defective(spark_plugs)).
recommended([jumper],distributor,poor(distributor_ground)).
recommended([visual_inspection],distributor,
                                              defective(distributor_rotor)).
recommended([hypothesis(moisture_on(distributor_cap))],
                                  distributor,moisture_on(distributor_cap)).
recommended(visual_inspection],distributor,
                                   cracked_or_tracked(distributor_cap)).
recommended([visual_inspection],distributor,
                          defective(distributor_wires_or_ignition_plugs)).
```

```
recommended(try_to_check],complete system,
                                                induction_firing_of_cyclinders).
recommended ([try_to_check],complete_system,
                                    incorrect(distributor_firing_sequence)).
recommended([test_light], complete_system,
                                                  incorrect(ignition_timing)).

preconditions(spark_test,sub_diagnosis,
                                                defective(ignition_system),[]).
preconditions(voltmeter_test1,sub_diagnosis,
                                    defective(primary_circuit_coil_side),
[not(diagnosis(defective(ignition_switch))),askif(has(jumper)),
                                      not(unable_to_use(voltmeter))]).
preconditions(voltmeter_test2,sub_diagnosis,
                                    defective(primary_circuit_coil_side)
not(diagnosis(defective ignition_switch))),askif(has(jumper)),
                                    not(unable_to_use(voltmeter))]).
preconditions(voltmeter,sub_diagnosis,
                          defective(primary_circuit_distributor_side),
                          [not(diagnosis(defective(ignition_switch))),
                  not(diagnosis(burned_or_damaged(ignition_points))),
                                      not(unable_to_use(voltmeter))]).
preconditions(visual_inspection,ignition_points
                                burned_or_damaged(ignition_points),[]).
preconditions(visual_inspection,ignition_points,
                                out_of_adjustment(ignition_points),[]).

preconditions(dwell_meter,ignition_points,
out_of_adjustment(ignition_points),[not(fact(engine_wont_start)),
                                      not(unable_to_use(dwell_meter))]).
preconditions(voltmeter,ballast_resistor,
  defective(ballast_resistor),[not(unable_to_use_(voltmeter))]).
preconditions(voltmeter,condenser,defective(condenser),
                                      [not(unable_to_use(voltmeter))]).
preconditions(electrical_test,ignition_switch,
          defective(ignition switch),[(askif(has(test_tamp_12V));
                                      not(unable_to_use(voltmeter)))]).
preconditions(voltmeter,ignition_coil,
                                    defective(coil_primary_resistance),
                                    [not(unable_to_use(voltmeter))]).
preconditions(ohmmeter,ignition_coil,
                                    defective(coil_secondary_resistance),
                                          [not(unable_to_use(ohmmeter))]).
preconditions(visual_inspection,plugs,defective(spark_plugs),[]).
preconditions(jumper,distributor,poor(distributor_ground),
                                                    [askif(has(jumper))]).
preconditions (visual_inspection,distributor,
                                      defective(distributor_rotor),[]).
preconditions(hypothesis(moisture_on(distributor_cap)),
                                distributor,moisture_on(distributor_cap),
                            [hypothesis(moisture_on(distributor_cap))]).
preconditions(visual_inspection,distributor,
                                    cracked_or_tracked(distributor_cap),[]).

preconditions(visual_inspection,distributor,
```

```
                    defective(distributor_wires_or_ignition_plugs),[]).
preconditions(try_to_check,complete_system,
                                        induction_firing_of_cylinders,[]).
preconditions(try_to_check,complete_system,
                                incorrect(distributor_firing_sequence),[]).
preconditions(test_light,complete_system,
                                            incorrect(ignition_timing).
                    [not(diagnosis(out_of_adjustment(ignition_points))),
                                        not(unable_to_use(test_light))]).

hr :- not(hypothesis(defective(spark_plugs))),fact(hard_starting),
fact(misfire_under_load),asserta(hypothesis(defective(spark_plugs))).
hr :- not(hypothesis(defective(spark_plugs))),fact(hard_starting),
fact(poor_pick_up),asserta(hypothesis(defective(spark_plugs))).
hr :- not(hypothesis(defective(spark_plugs))),
  askif(hear(detonation)),
  asserta(hypothesis(defective(spark_plugs))).
hr :-
  not(hypothesis(defective(distributor_wires_or_ignition_plugs))),
  fact(hard_starting),fact(poor_pick_up),
asserta(hypothesis(defective(distributor_wires_or_ignition_plugs))).
hr :- not(hypothesis(defective(condenser))),
  diagnosis(burned_or_damaged(ignition_points)),
  asserta(hypothesis(defective(condenser))).


hr :- not(hypothesis(induction_firing_of_cylinders)),
(fact(backfire_through_the_exhaust);fact(backfire_through_the_carburetor)),
fact(rough_idle),askif(hear(detonation)),
  asserta(hypothesis(induction_firing_of_cylinders)).
hr :- not(hypothesis(incorrect(distributor_firing_sequence))),
  fact(hard_starting),fact(backfire_through_the_carburetor),
  askif(recent_operator_job),
asserta(hypothesis(incorrect(distributor_firing_sequence))).
hr :- not(hypothesis(poor(distributor_ground))),
  fact(hard_starting),
  (fact(misfire_under_load);fact(engine_miss_at_high_rpm)),
  asserta(hypothesis(poor(distributor_ground))).
hr :- not(hypothesis(incorrect(ignition_timing))),
  askif(hear(detonation)),
  asserta(hypothesis(incorrect(ignition_timing))).

hr :- not(hypothesis(defective(coil_secondary_resistance))),
  fact(hard_starting),(fact(misfire_under_load);
  fact(engine_miss_at_high_rpm)),
  asserta(hypothesis(defective(coil_secondary_resistance))).
hr :- not(hypothesis(moisture_on(distributor_cap))),
  (fact(engine_wont_start);fact(hard_starting)),
  (askif(high_level_of_moisture_in_atmosphere);
  askif(wash_recently)),
  asserta(hypothesis(moisture_on(distributor_cap))).


hr :- not(hypothesis(cracked_or_tracked(distributor_cap))),
  fact(hard_starting),fact(susceptible_to_moisture),
```

```
  asserta(hypothesis(cracked_or_tracked(distributor_cap))).

hr :- not(defective(ignition_system)),fact(engine_wont_start),
  diagnosis(defective(ignition_system)),
  retract_sub_diagnosis(defective(ignition_system)),
  asserta(defective(ignition_system)).
hr :- not(hypothesis(defective(coil_secondary_resistance))),
  defective(ignition_system),
  asserta(hypothesis(defective(coil_secondary_resistance))).
hr :- not(defective(primary_circuit_coil_side)),
  defective(ignition_system),
  diagnosis(defective(primary_circuit_coil_side)),
  retract_sub_diagnosis(defective(primary_circuit_coil_side)),
  assert_hypothesis(defective(ballast_resistor)),
  assert_hypothesis(defective(coil_primary_resistance)),
  asserta(defective(primary_circuit_coil_side)).
hr :-not(defective(primary_circuit_distributor_side)),
  defective(ignition_system),
  diagnosis(defective(primary_circuit_distributor_side)),
  retract_sub_diagnosis(defective(primary_circuit_distributor_side)),
  assert_hypothesis(defective(condenser)),
  assert_hypothesis(out_of_adjustment(ignition_points)),
  assert_hypothesis(defective(distributor_rotor)),
  asserta(defective(primary_circuit_distributor_side)).

assert_hypothesis(Hypothesis) :- hypothesis(Hypothesis).
assert_hypothesis((Hypothesis) :- not(hypothesis(Hypothesis)),
                                  asserta(hypothesis(Hypothesis)).

retract_sub_diagnosis(Diagnosis) :- repeat,
                        retract(proved(diagnosis(Diagnosis,Equipment))),
                        not(proved(diagnosis(Diagnosis,Equip))),!.



questioncode(recent_operator_job,
$Has any operator job recently been done on ignition system$).
questioncode(high_level_of_moisture_in_atmosphere,
$Do you observe a high level of moisture in the atmosphere$).
questioncode(wash_recently,$Have you washed the car recently$).
questioncode(good_spark,$Is the spark good and consistent$).
questioncode(voltage_for_voltmeter_test1,$
With ignition switch on, did the voltmeter show 5.5 to 7 volt$).
questioncode(voltage_for_voltmeter_test2,
$While cranking, did the voltmeter show around 9 volts$).
questioncode(voltmeter_reading_remains_zero,$While  cranking,
        did the voltmeter reading remain zero or close to it$).

questioncode(burned_or_damaged(ignition_points),
$Did you observe burned or damaged ignition points$).
questioncode(out_of_adjustment(ignition_points),
$Did you note excessive open or close gap between points$).
questioncode(excessive_variation_in_dwell,
$Do you note an excessive variation in dwell,(over3 deg)
```

```
as the speed is increased$).
questioncode(zero_resistance,$Is the resistance zero$).
questioncode(shows_other_than_infinte,
$Did you note any reading other than infinite$).
questioncode(lamp_donot_flicker_but_light,$Did the lamp light$).
questioncode(read_1ohm_resistance,$Did you read about 1 ohm$).
questioncode(between4and8,$Is the reading between 4K and 8K ohm$).
questioncode(burned(spark_plug_points),
$Did you note burned spark plug points$).
questioncode(change_something,$Did it change something$).
questioncode(rotor_turns_appropriately,
$Does the rotor turn appropriately with no loosened components$).
questioncode(cracked_or_tracked(distributor_cap),
$Do you notice cracked or tracked distributor cap or component$).
questioncode(defective_ignition_wires,
$Do you observe any cracked, burned, or broken insulation$).
questioncode(induction_firing,$Do you note any consecutive wires
                                causing induction firing$).
questioncode(missing_cylinder,$Do you identify any missing one$).
questioncode(flash_light_intermittently,$Does the light flash intermittently$).
questioncode(unfixed_at_constant_engine_speed,
$Does the pointer appear to move on the index scale$).

code(ignition_points,
$Ignition points are located inside the distributor under the
distributor cap.$,$No more$).
code(condenser,$Condenser is located inside the distributor
under the distributor cap.$,$No more$).
code(ballast_resistor,$Ballast resistor lies between the ignition
coil and ignition switch.$,$No more$).
code(ignition_coil,
$Always located very close to engine.$,$No more$).
code(plugs,$Spark plugs are located on the engine connected to
distributor by ignition wires.$,$No more$).
code(distributor,$Distributor is one of the major components of the
ignition system.$,$Which is located  close to the engine.$).



code(1000,
$Check for spark at the coil high tension lead.$,
$Remove the coil high tension lead from the distributor and
position it approximately 1/4" from ground. Crank the engine
and observe the spark.$).
code(2000,$With engine at operating temperature, but stopped,
and the distributor side of the ignition coil grounded with a
jumper wire, hook up a voltmeter between the ignition coil
(switch side) and a good ground.$,$No more$).

code(3000,$With the voltmeter on the 16-20 volt scale, connect
one voltmeter lead to the distributor side of the coil. Remove
the high tension wire from the coil and ground it. Close ignition
switch and slowly bump the engine to open and close the points.$,
$No more$).
code(4000,$Visually inspect the ignition points.$,$For burned,
```

```
damaged, or out-of-adjustment points.$).
code(5000,
$Perform the dwell meter test according to manufacturer's
specifications.$,$No more$).
code(6000,
$Check the ballast resistor or resistor wire for an
open circuit, using an ohmmeter.$,$No more$).
code(7000,$Check the condenser for short.$,$Connect an ohmmeter
across the condenser body and the pigtail lead.$).
code(8000,$Check the ignition switch "on" position.$,
$Connect a jumper wire between the distributor side of the coil
and ground, and a 12V test lamp between the switch side of the
coil and ground. Remove the high tension lead from the coil. Turn
the igntion switch on and jiggle the key.$).
code(9000,$To check ignition coil resistance, primary side,
switch ohmmeter to low scale.  Connect the ohmmeter leads across
the primary terminals of the coil and read the low ohms scale.$,
$No more$).
code(9100,$Check the ignition coil secondary side resistance.$,
$Switch ohmmeter to high scale, connect one test lead to the
distributor cap end of the coil secondary cable, connect the
other test lead to the distributor terminal of the coil.$),
code(9200,$Remove the spark plugs.$,$Noting the cylinders from
which they were removed.$).
code(9300,$Connect a jumper wire between distributor body and a
good ground.$,$No more$).
code(9400,$Remove the distributor cap and check to make sure that
the rotor turns when the engine is cranked. Visually inspect the
distributor components.$,$No more$).
code(9500,$Inspect the distributor for cracked or tracked
distributor cap or components.$,$No more$).
code(9600,$Visually inspect the spark plug wires for cracking or
brittleness.$,$Spark plug wires can be checked visually by
bending them in a loop over your finger.$).
code(9700,$Ensure that no two wires are positioned so as to cause
induction firing,$,$Misfiring can be the result of spark plug
leads to adjacent, consecutively firing cylinders running parallel
and too close together.$).

code(9800,$Locate an ignition miss,$,$With the engine running,
remove each spark wire, one at a time, until one is found that
doesn't cause the engine to roughen and slow down.$).
code(9900,$Perform the ignition timing according to
manufacturer's specification.$,$No more$).

get_ready_equipment(spark_test,sub_diagnosis,
            defective(ignition_system)) :- code_interpreter(1000).

get_ready_equipment(Voltmeter_test,sub_diagnosis,
defective(primary_circuit_coil_side)) :- code_interpreter(2000).
get_ready_equipment(voltmeter,sub_diagnosis,
            defective(primary_circuit_distributor_side)) :-
                                                code_interpreter(3000).
get_ready_equipment(visual_inspection,ignition_points,
            burned_or_damaged(ignition_points)) :-
```

```
                                        code_interpreter(4000).
get_ready_equipment(visual_inspection,ignition_points,
        out_of_adjustment(ignition_points)) :-
                                        code_interpreter(4000).

get_ready_equipment(dwell_meter,ignition_points,
        out_of_adjustment(ignition_points)) :-
                                        code_interpreter(5000).
get_ready_equipment(voltmeter,ballast_resistor,
        defective(ballast_resistor)) :-
                                        code_interpreter(6000).
get_ready_equipment(voltmeter, condenser, defective(condenser)) :-
                                        code_interpreter(7000).
get_ready_equipment(electrical_test,ignition_switch,
        defective(ignition_switch)) :-
                                        code_interpreter(8000).

get_ready_equipment(voltmeter, ignition_coil,
                defective(coil_primary_resistance)) :-
                                        code_interpreter(9000).
get_ready_equipment(voltmeter, ignition_coil,
                defective(coil_secondary_resistance)) :-
                                        code_interpreter(9100).
get_ready_equipment(visual_inspection,plugs,
                defective(spark_plugs)) :-
                                        code_interpreter(9200).
get_ready_equipment(jumper,distributor,poor(distributor_ground))
                                        :- code_interpreter(9300).
get_ready_equipment(visual_inspection,distributor,
                                        defective(distributor_rotor)) :-
                                        code_interpreter(9400).
get_ready_equipment(hypothesis(moisture_on_distributor_cap)),
                distributor,moisture_on(distributor_cap)).
get_ready_equipment(visual_inspection,distributor,
cracked_or_tracked(distributor_cap)) :- code_interpreter(9500).
get_ready_equipment(visual_inspection,distributor,
                defective(distributor_wires_or_ignition_plugs))
                                        :- code_interpreter(9600).
get_ready_equipment(try_to_check,complete_system,
                induction_firing_of_cylinders) :-
                                        code_interpreter(9700).
get_ready_equipment(try_to_check,complete_system,
                incorrect(distributor_firing_sequence)) :-
                                        code_interpreter(9800).
get_ready_equipment(test_light,complete_system,
                        incorrect(ignition_timing)) :-
                                        code_interpreter(9900).



proved_diagnosis(spark_test,sub_diagnosis,
                        defective(ignition_system)) :-
                                        askifnot(good_spark).
proved_diagnosis(voltmeter_test1,sub_diagnosis,
                        defective(primary_circuit_coil_side)) :-
```

```
                                askifnot(voltage_for_voltmeter_test1).
proved_diagnosis(voltmeter_test2,sub_diagnosis,
                            defective(primary_circuit_coil_side)) :-
                                askifnot(voltage_for_voltmeter_test2).
proved_diagnosis(voltmeter,sub_diagnosis,
                    defective(primary_circuit_distributor_side)) :-
                                askifnot(voltmeter_reading_remains_zero).
proved_diagnosis(visual_inspection,ignition_points,
                               burned_or_damaged(ignition_points)) :-
                            askif(burned_or_damaged(ignition_points)).
proved_diagnosis(visual_inspection,ignition_points,
                            out_of_adjustment(ignition_points)) :-
                      assert_hypothesis(incorrect(ignition_timing)).
proved_diagnosis(dwell_meter,ignition_points,
                            out_of_adjustment(ignition_points)) :-
                                  askif(excessive_variation_in_dwell),
                  assert_hypothesis(defective(distributor_rotor)),
                assert_hypothesis(defective(coil_primary_resistance)),
           assert_hypothesis(defective(coil_secondary_resistance)),
                  assert_hypothesis(incorrect(ignition_timing)).
proved_diagnosis(voltmeter,ballast_resistor,
                                            defective(ballast_resistor)) :-
                                                askif(zero_resistance).
proved_diagnosis(voltmeter,condenser,defective(condenser)) :-
                                            askif(shows_other_than_infinite).
proved_diagnosis(electrical_test,ignition_switch,
                                            defective(ignition_switch)) :-
                                      askifnot(lamp_donot_flicker_but_light).
proved_diagnosis(voltmeter,ignition_coil,
                                      defective(coil_primary_resistance)) :-
                                            askif(read_1ohm_resistance).
proved_diagnosis(voltmeter,ignition_coil,
                            defective(coil_secondary_resistance)) :-
                                                askifnot(between4and8).
proved_diagnosis(visual_inspection,plugs,defective(spark_plugs))
                                     :- askif(burned(spark_plug_points)),
                                            assert_hypothesis(defective(condenser)).
proved_diagnosis(jumper,distributor,poor(distributor_ground)) :-
                                                askif(change_something).

proved_diagnosis(visual_inspection,distributor,
                                            defective(distributor_rotor)) :-
                                       askifnot(rotor_turns_appropriately).

proved_diagnosis(hypothesis(moisture_on(distributor_cap)),
                        distributor,moisture_on(distributor)cap)).

proved_diagnosis(visual_inspection,distributor,
                            cracked_or_tracked(distributor_cap)) :-
                       askif(cracked_or_tracked(distributor_cap)).
proved_diagnosis(visual_inspection,distributor,
                    defective(distributor_wires_or_ignition_plugs)) :-
                                            askif(defective_ignition_wires).
proved_diagnosis(try_to_check,complete_system,
                                            induction_firing_of_cylinders) :-
```

```
                                                    askif(induction_firing).
proved_diagnosis(try_to_check,complete_system,
                        incorrect(distributor_firing_sequence)) :-
                                                    askif(missing_cylinder).

proved_diagnosis(test_light,complete_system
                                                        incorrect(ignition_timing)) :-
                                                    askif(flash_light_intermittently),
                                assert_hypothesis(poor(distributor_ground)),
                    assert_hypothesis(cracked_or_tracked(distributor_cap)),
                         assert_hypothesis(defective(distributor_rotor)).
proved_diagnosis(test_light,complete_system,
                                                    incorrect(ignition_timing)) :-
                        askif(unfixed_at_constant_engine_speed),
                    assert_hypothesis(defective(distributor_rotor)).
```

%THE FOLLOWING CONTAINS THE CONTENTS OF ENGINE FILE.

```
list_of_expected_diagnosis(complete_system,
                           [high_resistance_in_engine]).
recommended([torque_test],complete_system,
                           high_resistance_in_engine).
preconditions(torque_test,complete_system,
                           high_resistance_in_engine,
                           [askif(has(torque_tool))]).
get_ready_equipment(torque_test,complete_system,
                           high_resistance_in_engine) :-
                           code_interpreter(10500).
proved_diagnosis(torque_test,complete_system,
                           high_resistance_in_engine) :-
                           askifnot(able_to_turn).

high_resistance_in_engine :- diagnosis(high_resistance_in_engine),
                           asserta(high_resistance_in_engine).

questioncode(able_to_turn,$Could you turn the engine freely$).
code(10500,$Attach the torque tool to front crank wheel of motor,
        and try to turn it by power.$,$No more$).
```

APPENDIX B

SAMPLE USER SESSIONS

/**********/

Actual responses given regarding the vehicle under consideration have been simulated throughout the sample consultations.

/**********/

CONSULTATION #1:

```
>api
Arity/Prolog Interpreter Version 4.0
Copyright (c) 1986 Arity Corporation
?-[global].

VEHICLE DIAGNOSIS EXPERT SYSTEM IMPLEMENTATION

Please enter "diagnosis" to start the consultation.

yes
?-diagnosis.

1:  Does the engine start at all?
2:  Does the engine run poorly?
Give numbers of questions whose answer is yes.[1].

1:  Does starter turn at all?
2:  Does starter turn, but not the engine?
3:  Does starter turn the engine very slowly?
4:  Does starter turn the engine normally?
Give numbers of questions whose answer is yes.[1].

1:  Does starter turn the engine very quickly?
2:  Does the engine fire intermittently?
3:  Does the engine fire consistently?
Give numbers of questions whose answer is yes.[ ].
```

*** BATTERY SECTION ***

Do you know where the battery is? (yes/no) ==> y.

Inspect the battery case.
Do you need further explanation? (yes/no) ==> n.

Is the battery case cracked? (yes/no) ==> n.
Is the battery case intact? (yes/no) ==> n.
Did the light come on when the knob turned on? (yes/no) ==> y.

Turn lights on high, try starter, and note action of lights.

Did the lights dim considerably or go out? (yes/no) ==> n.
Do you have a hydrometer? (yes/no) ==> y.
Do you know how to use a hydrometer? (yes/no) ==> y.

Test the state of charge of the battery using the hydrometer.

Does it indicate less than 1.140 @? (yes/no) ==> n.
Do you know where the battery cable connection is? (yes/no) ==> y.

Inspect the battery cables.
Do you need further explanation? (yes/no) ==> y.

For loose, broken, open cables and connections:
Did you notice bad cables or connections? (yes/no) ==> n.
Did you notice an open connection? (yes/no) ==> n.
Do you have a voltmeter? (yes/no) ==> y.
Do you know how to use a voltmeter? (yes/no) ==> y.

Connect prods of voltmeter on 3-volt scale to grounded battery post and starter motor housing. Close the starter switch and note the voltmeter reading.

Is the reading the same as the battery reading? (yes/no) ==> y.

MOST LIKELY DIAGNOSIS LIST:

LESS LIKELY DIAGNOSIS LIST:
    Open circuit

yes
?-

CONSULTATION #2:

```
>api
Arity/Prolog Interpreter Version 4.0
Copyright (c) 1986 Arity Corporation
?-[global].

VEHICLE DIAGNOSIS EXPERT SYSTEM IMPLEMENTATION

Please enter "diagnosis" to start the consultation.

yes
?-diagnosis.

1:  Does the engine start at all?
2:  Does the engine run poorly?
Give numbers of questions whose answer is yes.[1].

1:  Does starter turn at all?
2:  Does starter turn, but not the engine?
3:  Does starter turn the engine very slowly?
4:  Does starter turn the engine normally?
Give numbers of questions whose answer is yes.[3].

1:  Does starter turn the engine very quickly?
2:  Does the engine fire intermittently?
3:  Does the engine fire consistently?
Give numbers of questions whose answer is yes.[ ].

*** STARTER SECTION ***

*** LOADING BATTERY SECTION ***

Do you know where the battery is?                    (yes/no) ==> y.

Inspect the battery case.
Do you need further explanation?                     (yes/no) ==> n.

Is the battery case cracked?                         (yes/no) ==> n.
Is the battery case intact?                          (yes/no) ==> n.
Did the light come on when the knob turned on?       (yes/no) ==> y.

Turn lights on high, try starter, and note action of lights.
```

Did the lights dim considerably or go out? (yes/no) ==> y.
Did you hear a sound like a clattering? (yes/no) ==> n.
Do you have a hydrometer? (yes/no) ==> y.
Do you know how to use a hydrometer? (yes/no) ==> y.

Test the state of charge of the battery using the hydrometer.

Does it indicate less than 1.140 @? (yes/no) ==> n.

*** UNLOADING BATTERY SECTION ***

*** LOADING ENGINE SECTION ***

Attach the torque tool to the front crank wheel of the motor, and try to turn it by power.
Could you turn the engine freely? (yes/no) ==> y.

*** UNLOADING ENGINE SECTION ***

MOST LIKELY DIAGNOSIS LIST:

LESS LIKELY DIAGNOSIS LIST:
    Malfunction_in_starter_unit

yes
?-

CONSULTATION #3:

```
>api
Arity/Prolog Interpreter Version 4.0
Copyright (c) 1986 Arity Corporation
?-[global].

VEHICLE DIAGNOSIS EXPERT SYSTEM IMPLEMENTATION

Please enter "diagnosis" to start the consultation.

yes
?-diagnosis.

1:  Does the engine start at all?
2:  Does the engine run poorly?
Give numbers of questions whose answer is yes.[1].

1:  Does starter turn at all?
2:  Does starter turn, but not the engine?
3:  Does starter turn the engine very slowly?
4:  Does starter turn the engine normally?
Give numbers of questions whose answer is yes.[2].

1:  Does starter turn the engine very quickly?
2:  Does the engine fire intermittently?
3:  Does the engine fire consistently?
Give numbers of questions whose answer is yes.[2].

I found a contradiction!
Check and repeat your answer.
1:  Does starter turn at all?
2:  Does starter turn, but not the engine?
3:  Does starter turn the engine very slowly?
4:  Does starter turn the engine normally?
Give numbers of questions whose answer is yes.[2].

1:  Does starter turn the engine very quickly?
2:  Does the engine fire intermittently?
3:  Does the engine fire consistently?
Give numbers of questions whose answer is yes.[ ].
```

*** STARTER SECTION ***

Is that your problem:
Starter spins free but won't engage? (yes/no) ==> n.

*** LOADING ENGINE SECTION ***

Do you have a torque tool? (yes/no) ==> y.
Attach the torque tool to the front crank wheel of the motor, and try to turn it by power.
Could you turn the engine freely? (yes/no) ==> n.

*** UNLOADING ENGINE SECTION ***

MOST LIKELY DIAGNOSIS LIST:

LESS LIKELY DIAGNOSIS LIST:
    High-resistance-in-engine

yes
?-

CONSULTATION #4:

```
>api
Arity/Prolog Interpreter Version 4.0
Copyright (c) 1986 Arity Corporation
?-[global].

VEHICLE DIAGNOSIS EXPERT SYSTEM IMPLEMENTATION

Please enter "diagnosis" to start the consultation.

yes
?-diagnosis.

1:  Does the engine start at all?
2:  Does the engine run poorly?
Give numbers of questions whose answer is yes.[1].

1:  Does starter turn at all?
2:  Does starter turn, but not the engine?
3:  Does starter turn the engine very slowly?
4:  Does starter turn the engine normally?
Give numbers of questions whose answer is yes.[4].

1:  Does starter turn the engine very quickly?
2:  Does the engine fire intermittently?
3:  Does the engine fire consistently?
Give numbers of questions whose answer is yes.[2].

*** IGNITION SECTION ***

Did you hear a sound like a detonation?                (yes/no) ==> n.
Do you observe a high level of moisture in the
     atmosphere?                                       (yes/no) ==> n.
Have you washed the car recently?                      (yes/no) ==> n.

Check for spark at the coil high tension lead.
Do you need further explanation?                       (yes/no) ==> y.

Remove the coil high tension lead from the distributor and position it
approximately 1/4" from the ground.  Crank the engine and observe
the spark.
Is the spark good and consistent?                      (yes/no) ==> n.

Do you have a test-lamp-12V?                           (yes/no) ==> y.
```

Check the ignition switch "on" position.
Do you need further explanation? (yes/no) ==> y.

Connect a jumper wire between the distributor side of the coil and ground, and a 12V test lamp between the switch side of the coil and ground. Remove the high tension lead from the side coil. Turn the ignition switch on and jiggle the key.
Did the lamp light? (yes/no) ==> n.

Do you have an ohmmeter? (yes/no) ==> y.
Do you know how to use an ohmmeter? (yes/no) ==> y.
Do you know where the ignition-coil is? (yes/no) ==> y.

Check the ignition coil secondary side resistance.
Do you need further explanation? (yes/no) ==> y.

Switch ohmmeter to high scale, connect one test lead to the distributor cap end of the coil secondary cable, connect the other test lead to the distributor terminal of the coil.
Is the reading between 4K and 8K ohm? (yes/no) ==> n.

MOST LIKELY DIAGNOSIS LIST:

LESS LIKELY DIAGNOSIS LIST:
    defective(coil_secondary_resistance)
    defective(ignition_switch)

yes
?-

CONSULTATION #5:

```
>api
Arity/Prolog Interpreter Version 4.0
Copyright (c) 1986 Arity Corporation
?-[global].

VEHICLE DIAGNOSIS EXPERT SYSTEM IMPLEMENTATION

Please enter "diagnosis" to start the consultation.

yes
?-diagnosis.

1:  Does the engine start at all?
2:  Does the engine run poorly?
Give numbers of questions whose answer is yes.[1,2].

I found a contradiction!
Check and repeat your answer
1:  Does the engine start at all?
2:  Does the engine run poorly?
Give numbers of questions whose answer is yes.[2].

1:  Do you have a hard starting problem?
2:  Do you have a rough idle?
3:  Do you have stalling?
4:  Does the engine die at high speed?
Give numbers of questions whose answer is yes.[1,2,4].

1:  Do you have hesitation (on acceleration from a standing stop)?
2:  Do you have poor pickup?
3:  Do you have lack of power?
4:  Do you have backfire through the carburetor?
Give numbers of questions whose answer is yes.[2].

1:  Do you have backfire through the exhaust?
2:  Do you have blue exhaust gases?
3:  Do you have black exhaust gases?
4:  Do you have running on (after the ignition is shut off)?
Give numbers of questions whose answer is yes.[ ].
```

1: Is it susceptible to moisture?
2: Does the engine misfire under load?
3: Does the engine misfire at speed?
4: Does the engine misfire at idle speed?
Give numbers of questions whose answer is yes.[3].

*** IGNITION SECTION ***

Did you hear a sound like a detonation? (yes/no) ==> n.
Do you observe a high level of moisture in the
atmosphere? (yes/no) ==> n.
Have you washed the car recently? (yes/no) ==> n.
Do you have a voltmeter? (yes/no) ==> y.
Do you know how to use a voltmeter? (yes/no) ==> y.

Check the ignition coil secondary side resistance.
Do you need further explanation? (yes/no) ==> y.

Switch ohmmeter to high scale, connect one test lead to the distributor cap end of the coil secondary cable, connect the other test lead to the distributor terminal of the coil.
Is the reading between 4K and 8K ohm? (yes/no) ==> n.

MOST LIKELY DIAGNOSIS LIST:

LESS LIKELY DIAGNOSIS LIST:
defective(coil_secondary_resistance)

yes
?-

LIST OF REFERENCES

1. Waterman, D. A., A Guide to Expert Systems, Addison-Wesley Publishing Company, Inc., 1986.

2. Buchanan, B. G., and Shortliffe, E. H., Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Addison-Wesley Publishing Company, Inc., 1984.

3. Arity/Prolog Version 4.0 Copyright 1986, CSA Press, Hudson, MA.

4. Forsyth, R., Expert Systems: Principles and case studies, Chapman and Hall, 1984.

5. Rowe, N., Introduction to Artificial Intelligence Through Prolog, Prentice-Hall, 1988.

6. Reiter, R., A theory of Diagnosis from First Principles, Artificial Intelligence, V. 32, 1987.

7. Kleer, J., and Williams, B. C., Diagnosing Multiple Faults, Artificial Intelligence, V. 32, 1987.

8. Alexander, J. H., Freiling, M. J., Troubleshooting with the Help of an Expert System, Technical Report No. CR-85-05, Artificial Intelligence Dept, Computer Research Laboratory, Tektronix, Inc., August 1984.

9. Chilton's Repair & Tune-up Guide: Chevy II and Nova 1962-79, Chilton Book Company, 1979.

10. Chilton's Auto Repair Manual 1964-1971, Chilton Book Company, 1971.

11. Clocksin, W. F., Mellish, C. S., Programming in Prolog, Springer-Verlag, 1984.

BIBLIOGRAPHY

1. Sterling, L., Shapiro, E., The Art of Prolog, The MIT Press, 1986.

2. Addis, T. R., Expert Systems: An Evolution in Information Retrieval, International Computers Limited, Technical Journal, May 1980.

3. Addis, T. R., Towards an 'Expert' Diagnostic System, ICL Technical Journal, May 1980.

4. Davis, R., Diagnostic Reasoning Based on Structure and Behavior, Artificial Intelligence, V. 24, 1984.

5. Gensereth, M. R., The Use of Design Descriptions in Automated Diagnosis, Artificial Intelligence, V. 24, 1984.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defence Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	1
4.	Associate Professor N. C. Rowe Code 52Rp Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	1
5.	Professor Robert McGhee Code 52Mz Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	1
6.	Professor Yuh-jeng Lee Code 52Le Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	1
7.	LTJG Mufit Can Selek, Turkish Navy Sogutlu cesme, Elmali cesme sok. Huzur apt. Dai: 6 Kadikoy - Istanbul, Turkey	2
8.	Deniz Harp Okulu Kitapligi Deniz Harp Okulu Komutanligi Tuzla - Istanbul, Turkey	1

Thesis
S41253 Selek
c.1 An expert system for the diagnosis of vehicle malfunctions.