Expert System for Automobile Industry

A Dissertation Submitted in partial fulfilment for the award of the degree of

Master of Engineering

in
Computer Science

Submitted By
Harsh K. Sadawarti

Department of Computer Science & Engineering
Thapar Institute of Engineering & Technology
(Deemed University)
Patiala - 147004

2000

Dedicated

to my beloved son

Saarthak

Certificate

I hereby certify that the work presented in the thesis entitled "**Expert System for Automobile Industry**" in fulfilment of the requirement for the award of the degree of master of Engineering in Computer Science is being submitted in the Department of Computer Science and Engineering. It is further certified that the work carried out for the thesis is an authentic record done under the supervision of Dr. P. K. Bansal.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other university.

(Harsh K. Sadawarti)

This is to certify that the above statement made by the candidate is correct to the best of our knowledge.

28/5/2000

(Dr. P. K. Bansal)
Professor
Department of Computer Science and Engineering
TIET, Patiala.

29.5.2000

Head
Department of Computer Science and Engineering
TIET, Patiala.

Dean
Academic Affairs
TIET, Patiala

Acknowledgement

Before I get into the thick of things I would like to add a few heart felt words for the people who were part of this thesis in numerous ways, who gave unending support right from the stage the idea was conceived.

I would like to thank Thapar Institute of Engineering & Technology, Patiala for providing me an opportunity to carry out the thesis on **"Expert System for Automobile Industry"** as a part of ME degree program.

I consider it a great privilege to express my deep sense of gratitude to Dr. P.K. Bansal, for his invaluable guidance, constant inspiration and critical review throughout the course of this work. He has always been more that generous in sparing his very valuable time.

I am also thankful to all my fellow friends Amanpreet Sethi, Oberoi, Jasbir Ratol and my student Ajay Pal Singh for their time to time suggestions and cooperation.

There are times in such projects when the clock beats you time and again and you just run out of energy and just want to finish it off once and for all. My sincere appreciation goes to my wife Ritcha for her cooperation and untiring patience in many odd hours during the thesis work, she made me endure such times with her unfailing humour and warm wishes. And finally thanks to my parents for their encouragement and moral support.

Harsh K. Sadawarti

Abstract

During the last century, civilization has undergone a knowledge explosion, and it has become impractical if not impossible for an individual to gain ever emerging area of human endeavor. We are a species of specialists. Similarly, intelligent computer systems must also specialize. Instead of trying to design a computer system capable of expertise in several fields, the focus must be on a single, carefully chosen, clearly defined subject.

Books and manuals have a tremendous amount of knowledge but a human being has to read and interpret the knowledge for it to be used. Thus to exploit the available wisdom, knowledge based systems have been devised. Knowledge-based systems collect the small fragments of human know-how into a knowledge base which is used to reason through a specific problem area, with the help of a computer, using the knowledge that is appropriate for that problem domain.

The objective of the thesis "Expert System for Automobile Industry" is to develope an expert system for Automobile Problem Diagnostics and Maintenance. The objective of the expert system is to identify the problem in an automobile from the information provided by the user and the information stored in the knowledge base of the system. Finding and diagnosing problems in Automobiles is not an easy task. Everyday new models are being released in the market. It is difficult to keep track of each and every model and the problems occurring in each of them for a human being even if he is an expert in the field. In the domain of Automobiles, the problems, to a great extent, are very much common but genuine experts in the field of automobiles are not so common, or not so easily accessible. Thus use of a computer system to diagnose and suggest solutions is the need of the hour.

Expert systems are the end products of knowledge engineers. To build an expert system a knowledge engineer starts by reading domain-related literature to become familiar with the issues and the terminology, with that as a foundation, the knowledge engineer then holds interviews with one or more domain experts to "acquire" their knowledge. Human knowledge is immense and it is impossible, with the computing resources available with us today, to capture all the available knowledge. The expert system developed in the thesis captures the small fragments of human know-how about "automobile related problems and their solutions" into a knowledge base that is used to reason through a problem, with the help of a computer. Conventional programming languages, such as FORTRAN and C/C++, are designed and optimized for the procedural manipulation of data (such as numbers and arrays). Humans, however, often solve complex problems using very abstract, symbolic approaches which are not well suited for implementation in conventional languages.

Prolog uses heuristics to solve problems. It allows representing knowledge in an abstract form, which is closer to the human thinking procedure. The expert system has been developed using Prolog. The knowledge base of the expert system consists of rules of thumb that are used to assert the authenticity of a problem.

In this thesis, as a knowledge engineer, the knowledge from the domain of "Automobiles" has been organized in the expert system's knowledge base. The knowledge gathered from the experts and literature related to automobiles has been translated into software that a computer can use to solve problems. The knowledge of the automobile experts has been abstracted into an objective form for the database of the expert system, which is used to diagnose and resolve problems related to the automobiles. And this knowledge base can be extended to include new problem domains.

The expert system is quite easy to use, the interface is menu driven and uncomplicated. The user responds to lucid questions, in YES or NO only, as queried by the system. The answers to these questions help the expert system to locate the solution to the problem encountered by the user of the expert system. Those people, who have to deal with an automobile everyday, or frequently, like a mechanic, automobile owner, engineers, service personnel, workshops etc can find extensive use of this system.

Table of Contents

Table of Figures

Chapter 1

Introduction

Introduction

In Knowledge Lies the Power. This realization led to the development of knowledge based systems. These systems, which use specialized sets of, coded knowledge to "reason" and perform intelligent tasks. This is in contrast with more conventional type programs, which rely on data and general algorithms (weak methods) to solve less intelligent tasks. Knowledge based systems are much more successful in most areas than the conventional programs. The ability of these systems to explain the reasoning process through back-traces and to handle levels of confidence and uncertainty provides an additional feature that conventional programming doesn't handle.

1.1 Artificial Intelligence

It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable.

After W.W.II, a number of people independently started to work on intelligent machines. The English mathematician Alan Turing may have been the first. He gave a lecture on it in 1947. He also may have been the first to decide that AI was best researched by programming computers rather than by building machines. By the late 1950s, there were many researchers on AI, and most of them were basing their work on programming computers.

Artificial Intelligence is the study of how to make computers do things, which at the moment people do better. This definition is somewhat ephemeral, because of its reference to the current state of computer science. And it fails to include some areas of potentially very large impact, like problems that cannot now be solved by well either by people or computers.

1.2 Definition of AI

1. **From the perspective of intelligence:** Artificial intelligence is making machines "intelligent" - acting, as we would expect people to act.

2. **From a research perspective:** Artificial intelligence is the study of how to make computers do things which, at the moment, people do better.

3. **From a business perspective:** AI is a set of very powerful tools, and methodologies for using those tools to solve business problems.

4. **From a programming perspective:** AI includes the study of symbolic programming, problem solving, and search.

5. **From a layman's point of view:** AI can be defined as the attempt to get real machines to behave like the ones in the movies.

6. **Another definition of AI:** AI, according to Hollywood, is going to take over the world and enslave us all!

1.3 What is Intelligent

Intelligence is the computational part of the ability to achieve goals in the world. Varying kinds and degrees of intelligence occur in people, many animals and some machines. There isn't a solid definition of intelligence that doesn't depend on relating it to human intelligence. The problem is that we cannot yet characterize in general what kinds of computational procedures we want to call intelligent. We understand some of the mechanisms of intelligence and not others.

Intelligence involves mechanisms, and AI research has discovered how to make computers carry out some of them and not others. If doing a task requires only mechanisms that are well understood today, computer programs can give very impressive performances on these tasks. Such programs should be considered ``somewhat intelligent".

1.3.1 Is AI about simulating human intelligence?

Sometimes this is true, but not always or even usually. On the one hand, we can learn something about how to make machines solve problems by observing other people or just by observing our own methods. On the other hand, most work in AI involves studying the problems the world presents to intelligence rather than studying people or animals. AI researchers are free to use methods that are not observed in people or that involve much more computing than people can do.

1.3.2 Turing test

Alan Turing's 1950 article Computing Machinery and Intelligence discussed conditions for considering a machine to be intelligent. He argued that if the machine could successfully pretend to be human to a knowledgeable observer then you certainly should consider it intelligent. This test would satisfy most people but not all philosophers. The observer could interact with the machine and a human by Teletype (to avoid requiring that the machine imitate the appearance or voice of the person), and the human would try to persuade the observer that it was human and the machine would try to fool the observer. The Turing test is a one-sided test. A machine that passes the test should certainly be considered intelligent, but a machine could still be considered intelligent without knowing enough about humans to imitate a human.

The Turing test and the various partial Turing tests that have been implemented, i.e. with restrictions on the observer's knowledge of AI and the subject matter of questioning. It turns out that some people are easily led into believing that a rather dumb program is intelligent.

1.4 Domains of AI

AI began in the early 1960s -- the first attempts were game playing (checkers), theorem proving (a few simple theorems) and general problem solving (only very simple tasks). General problem solving was much more difficult than originally anticipated. Researchers were unable to tackle problems routinely handled by human experts. The name "artificial intelligence" came from the roots of the area of study.

AI researchers are active in a variety of domains

- Formal Tasks (mathematics, games)
- Mundane tasks (perception, robotics, natural language, common sense reasoning)
- Expert tasks (financial analysis, medical diagnostics, engineering, scientific analysis, and other areas)

Typically AI programs focus on symbols rather than numeric processing Prolog and other tools for AI are less efficient in numerical processing as compared to procedural languages. They are more suitable for their ability to infer facts and conclusions from other facts.

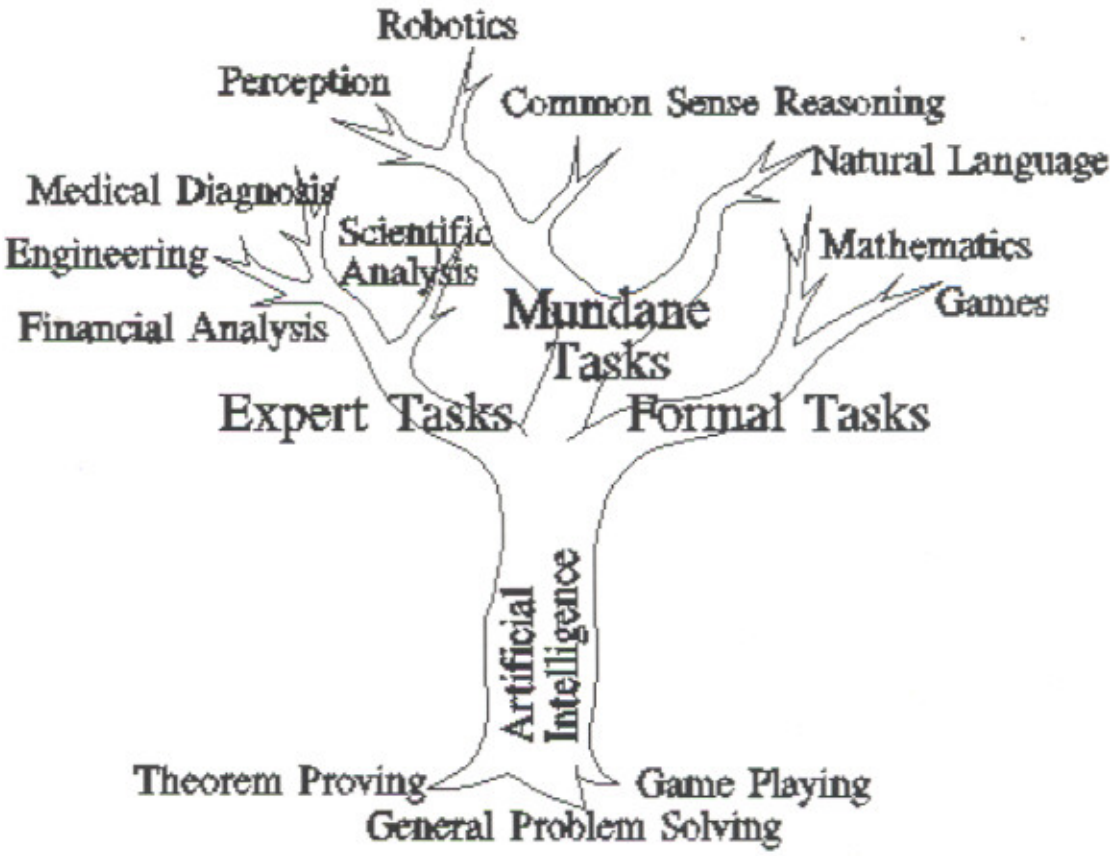

Fig 1.1 Task Domain of Artificial Intelligence

1.5 Branches of AI

Artificial Intelligence is being used in many fields, here's a list, but some branches are surely missing, because no one has identified them yet. Some of these may be regarded as concepts or topics rather than full branches.

1.5.1 Logical AI

What a program knows about the world in general the facts of the specific situation in which it must act, and entrances of some mathematical logical language represent all its goals. The program decides what to do by inferring that certain actions are appropriate for achieving its goals.

1.5.2 Search

AI programs often examine large numbers of possibilities, e.g. moves in a chess game or inferences by a theorem-proving program. Discoveries are continually made about how to do this more efficiently in various domains.

1.5.3 Pattern recognition

When a program makes observations of some kind, it is often programmed to compare what it sees with a pattern. For example, a vision program may try to match a pattern of eyes and a nose in a scene in order to find a face. More complex patterns, e.g. in a natural language text, in a chess position, or in the history of some event are also studied. These more complex patterns require quite different methods than do the simple patterns that have been studied the most.

1.5.4 Representation

Facts about the world have to be represented in some way. Usually languages of mathematical logic are used.

1.5.5 Inference

From some facts, others can be inferred. Mathematical logical deduction is adequate for some purposes, but new methods of non-monotonic inference have been added to logic since the 1970s. The simplest kind of non-monotonic reasoning is default reasoning in which a conclusion is to be inferred by default, but the conclusion can be withdrawn if there is evidence to the contrary. For example, when we hear of a bird, we infer that it can fly, but this conclusion can be reversed when we hear that it is a penguin. It is the possibility that a conclusion may have to be withdrawn that constitutes the non-monotonic character of the reasoning. Ordinary logical reasoning is monotonic in that the set of conclusions that can the drawn from a set of premises is a monotonic increasing function, of the premise common sense knowledge and reasoning. This is the area in which AI is farthest from human-level, in spite of the fact that it has been an active research area since the 1950s. While there has been considerable progress, e.g. in developing systems of non-monotonic reasoning and theories of action, yet more new ideas are needed.

1.5.6 Learning from experience

Programs do that. The approaches to AI based on connectionism and neural nets specialize in that. There is also learning of laws expressed in logic. Programs can only learn what facts or behaviours their formalisms can represent, and unfortunately learning systems are almost all based on very limited abilities to represent information.

1.5.7 Planning

Planning programs start with general facts about the world (especially facts about the effects of actions), facts about the particular situation and a

statement of a goal. From these, they generate a strategy for achieving the goal. In the most common cases, the strategy is just a sequence of actions.

1.5.8 Epistemology

This is a study of the kinds of knowledge that are required for solving problems in the world.

1.5.9 Ontology

Ontology is the study of the kinds of things that exist. In AI, the programs and sentences deal with various kinds of objects, and we study what these kinds are and what their basic properties are. Emphasis on ontology begins in the 1990s.

1.5.10 Heuristics

A heuristic is a way of trying to discover something or an idea imbedded in a program. The term is used variously in AI. Heuristic functions are used in some approaches to search to measure how far a node in a search tree seems to be from a goal. Heuristic predicates that compare two nodes in a search tree to see if one is better than the other, i.e. constitutes an advance toward the goal, may be more useful.

1.5.11 Genetic programming

Genetic programming is a technique for getting programs to solve a task by mating random Lisp programs and selecting fittest in millions of generations.

1.6 Applications of AI

1.6.1 Game playing

You can buy machines that can play master level chess for a few hundred dollars. There is some AI in them, but they play well against people mainly through brute force computation--looking at hundreds of thousands of positions. To beat a world champion by brute force and known reliable heuristics requires being able to look at 200 million positions per second.

1.6.2 Speech recognition

In the 1990s, computer speech recognition reached a practical level for limited purposes. Thus United Airlines has replaced its keyboard tree for flight information by a system using speech recognition of flight numbers and city names. It is quite convenient. On the other hand, while it is possible to instruct some computers using speech, most users have gone back to the keyboard and the mouse as still more convenient.

1.6.3 Understanding natural language

Just getting a sequence of words into a computer is not enough. Parsing sentences is not enough either. The computer has to be provided with an

understanding of the domain the text is about, and this is presently possible only for very limited domains.

1.6.4 Computer vision

The world is composed of three-dimensional objects, but the inputs to the human eye and computers' TV cameras are two-dimensional. Some useful programs can work solely in two dimensions, but full computer vision requires partial three-dimensional information that is not just a set of two-dimensional views. At present there are only limited ways of representing three-dimensional information directly, and they are not as good as what humans evidently use.

1.6.5 Expert systems

A "knowledge engineer" interviews experts in a certain domain and tries to embody their knowledge in a computer program for carrying out some task. How well this works depends on whether the intellectual mechanisms required for the task are within the present state of AI. When this turned out not to be so, there were many disappointing results. One of the first expert systems was MYCIN in 1974, which diagnosed bacterial infections of the blood and suggested treatments. It did better than medical students or practicing doctors, provided its limitations were observed. Namely, its ontology included bacteria, symptoms, and treatments and did not include patients, doctors, hospitals, death, recovery, and events occurring in time. Its interactions depended on a single patient being considered. Since the experts consulted by the knowledge engineers knew about patients, doctors, death, recovery, etc., it is clear that the knowledge engineers forced what the experts told them into a predetermined framework. In the present state of AI, this has to be true. The usefulness of current expert systems depends on their users having common sense.

1.6.7 Heuristic classification

One of the most feasible kinds of expert system given the present knowledge of AI is to put some information in one of a fixed set of categories using several sources of information. An example is advising whether to accept a proposed credit card purchase. Information is available about the owner of the credit card, his record of payment and also about the item he is buying and about the establishment from which he is buying it (e.g., about whether there have been previous credit card frauds at this establishment).

1.7 Tools for AI

AI programming languages include:

1.7.1 LISP

LISP, developed in the 1950s, is the early programming language strongly associated with AI. LISP is a functional programming language with

procedural extensions. LISP (LISt Processor) was specifically designed for processing heterogeneous lists -- typically a list of symbols. Features of LISP that made it attractive to AI researchers included run- time type checking, higher order functions (functions that have other functions as parameters), automatic memory management (garbage collection) and an interactive environment.

1.7.2 PROLOG

The second language strongly associated with AI is PROLOG. PROLOG was developed in the 1970s. PROLOG is based on first order logic. PROLOG is declarative in nature and has facilities for explicitly limiting the search space.

1.7.3 Object-oriented languages

Object-oriented languages are a class of languages more recently used for AI programming. Important features of object-oriented languages include:

- Concepts of objects and messages
- Objects bundle data and methods for manipulating the data
- Sender specifies what is to be done, receiver decides how to do it.
- Inheritance (object hierarchy where objects inherit the attributes of the more general class of objects)

Examples of object-oriented languages are Smalltalk, Objective C, C++. Object oriented extensions to LISP (CLOS - Common LISP Object System) and PROLOG (L&O - Logic & Objects) are also used.

1.8 Formulation of The Problem

There are many automobiles (4-wheeled motor vehicle; usually propelled by an internal combustion engine) in use today. It has almost become an automobile revolution, with almost ever person using some kind of automobile, whether it's a personal or public vehicle like a Bus or a Car. And people usually face problem while using them. In today's fast world no one has enough time to consult an Expert Automobile Engineer/ Mechanic for common problems.

Common problems like problems with battery, engine, which the owners/ users of the automobile can handle by themselves, can be diagnosed using the expert system, which has been developed. Expert in Automobile Maintenance can take help of this system for quick diagnostic and removal of problems from the automobile. Thus saving both money and time of users and experts in the field of Automobile Maintenance.

1.9 Organisation of The Thesis

The thesis has been organised as follows:

The field of Artificial Intelligence, relevance, and its applications has been discussed in the first chapter.

Expert Systems and their structure, their organization, types, tools for expert systems have been described in the second chapter.

Prolog the Programming Language of choice for building Expert Systems and solving other AI problems has been illustrated in the third chapter.

The Fourth Chapter describes the Expert System for "**Automobile Problem Diagnostics and Maintenance**".

The Fifth Chapter concludes the thesis with the future scope of the Expert Systems and their relevance in our day to day life.

1.10 Conclusion

The goal in AI is to construct working programs that solve problems we are interested in. But the mechanisms of brains are very different in detail from those in Computers, even though they may be doing similar sorts of things (sorting, transforming, and using information).

Some people think that it will never be possible to understand and replicate all the important aspects of brain function unless we replace computers with new kinds of machines, or perhaps build hybrid machines using different technologies. This conclusion is premature. There are two reasons:

- We do not yet know what the real potential of computers will turn out to be as we develop new types, which are faster and smaller and can be linked together in vast collections of cooperating systems.

- There is much that we do not know about brains: including what they do and how they do it.

So it cannot yet be said with confidence that there's ANYTHING brains can do which computers will NEVER be able to do, even though there are many things brains can do which existing computers cannot do (and vice versa!).

Chapter 2

Expert Systems

Introduction

Within the last ten years, artificial intelligence-based computer programs called expert systems have received a great deal of attention. The reason for all the attention is that these programs have been used to solve an impressive array of problems in a variety of fields. Well-known examples include computer system design, locomotive repair, and gene cloning.

An expert system stores the knowledge of one or more human experts in a particular field. The field is called a domain. The experts are called domain experts. A user presents the expert system with the specifics of a problem within the domain. The system applies its stored knowledge to solve the problem. Knowledge-based expert systems, or simply expert systems, use human knowledge to solve problems that normally would require human intelligence. These expert systems represent the expertise knowledge as data or rules within the computer. These rules and data can be called upon when needed to solve problems. Books and manuals have a tremendous amount of knowledge but a human has to read and interpret the knowledge for it to be used. Conventional computer programs perform tasks using conventional decision-making logic -- containing little knowledge other than the basic algorithm for solving that specific problem and the necessary boundary conditions. This program knowledge is often embedded as part of the programming code, so that as the knowledge changes, the program has to be changed and then rebuilt.

Knowledge-based systems collect the small fragments of human know-how into a knowledge base which is used to reason through a problem, using the knowledge that is appropriate. A different problem, within the domain of the knowledge base, can be solved using the same program without reprogramming. The ability of these systems to explain the reasoning process through back-traces and to handle levels of confidence and uncertainty provides an additional feature that conventional programming doesn't handle.

2.1 What are Expert Systems (ES)?

Definitions of expert systems vary. Some definitions are based on function. Some definitions are based on structure. Some definitions have both functional and structural components. Many early definitions assume rule-based reasoning.

2.1.1 Definition on functional components

1. **What the system does (rather than how):** A computer program that behaves like a human expert in some useful ways.
2. **Problem area:** Solve problems efficiently and effectively in a narrow problem area. Typically, pertains to problems that can be symbolically represented.
3. **Problem difficulty:** Apply expert knowledge to difficult real world problems. Solve problems that are difficult enough to require significant human expertise for their solution, address problems normally thought to require human specialists for their solution.
4. **Performance requirement:** The ability to perform at the level of an expert. Programs that mimic the advice-giving capabilities of human experts, matches a competent level of human expertise in a particular field. can offer intelligent advice or make an intelligent decision about a processing function, allows a user to access this expertise in a way similar to that in which he might consult a human expert, with a similar result.
5. **Explain reasoning:** The capability of the system, on demand, to justify its own line of reasoning in a manner directly intelligible to the enquirer, incorporation of explanation processes .

2.1.2 Definition on structural components

How the system functions:

1. **Use AI techniques:** Using the programming techniques of artificial intelligence, especially those techniques developed for problem solving.

2. **Knowledge component:** The embodiment within a computer of a knowledge-based component, from an expert skill, a computer based system in which representations of expertise are stored. The knowledge of an expert system consists of facts and heuristics. The 'facts' constitute a body of information that is widely shared, publicly available, and generally agreed upon by experts in the field. Expert systems are sophisticated computer programs that manipulate knowledge to solve problems.

3. **Separate knowledge and control:** Make domain knowledge explicit and separate from the rest of the system.

4. **Use inference procedures (heuristics and uncertainty):** An intelligent computer program that uses knowledge and inference procedures. The style adopted to attain these characteristics is rule-based programming. Exhibit intelligent behavior by skillful application of heuristics. The heuristics are mostly private, little rules of good judgment (rules of plausible reasoning, rules of good guessing) that characterize expert-level decision making in the field. Incorporation of ways of handling uncertainty.

5. **Model human expert:** This can be thought of as a model of the expertise of the best practitioners of the field, representation of domain-specific knowledge in the manner in which the expert thinks, involving the use of appropriate information acquired previously from human experts.

2.2 EXPERT SYSTEM COMPONENTS

The part of the expert system that stores the knowledge is called the knowledge base. The part that holds the specifics of the to-be-solved problem are (somewhat misleadingly) called the global database (think of it as a kind of "scratch pad"). The part that applies the knowledge to the problem is called the inference engine.

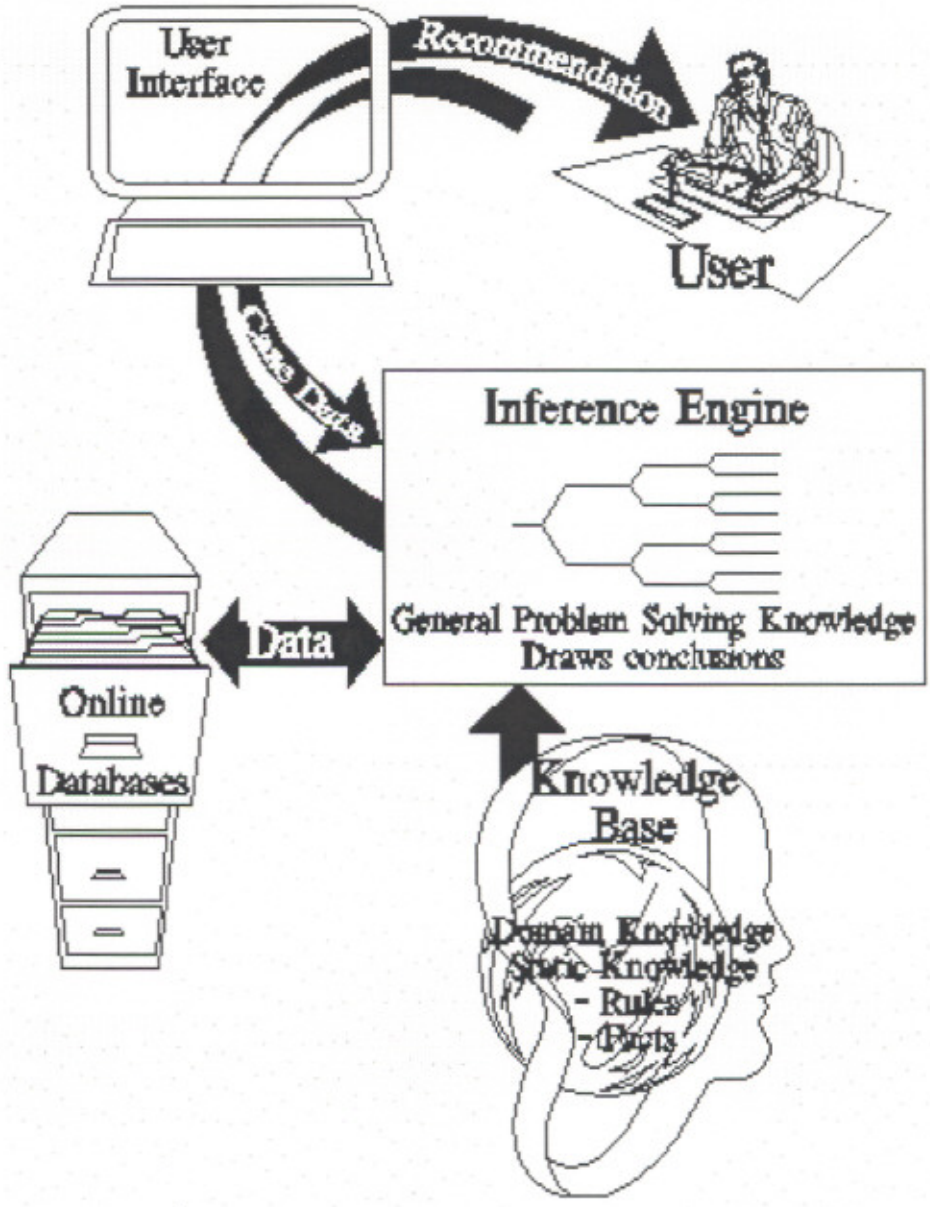

Fig 2.2 Structure of an Expert System

A ``knowledge engineer'' interviews experts in a certain domain and tries to embody their knowledge in a computer program for carrying out some task. One of the first expert systems was MYCIN in 1974, which diagnosed bacterial infections of the blood and suggested treatments. It did better than medical students or practicing doctors, provided its limitations were observed. Namely, its ontology included bacteria, symptoms, and treatments and did not include patients, doctors, hospitals, death, recovery, and events occurring in time. Its interactions depended on a single patient being considered. Since the experts consulted by the knowledge engineers knew about patients, doctors, death, recovery, etc., it is clear that the knowledge engineers forced what the experts told them into a predetermined framework. In the present state of AI, this has to be true. The usefulness of current expert systems depends on their users having common sense.

As is the case with most contemporary computer programs, expert systems typically have friendly user interfaces. A friendly interface doesn't make the system's internals work any more smoothly, but it does enable inexperienced users to specify problems for the system to solve and to understand the system's conclusions.

The various components are:

2.2.1 User Interface

- Friendly
- Maybe "Intelligent"
    - Knowledge of how to present information
    - Knowledge of user preferences...possibly accumulate with use

2.2.2 Databases

- Contains some of the data of interest to the system
- May be connected to on-line company or public database
- Human user may be considered a database

2.2.3 Inference Engine

- General problem-solving knowledge or methods
- Interpreter analyzes and processes the rules
- Scheduler determines which rule to look at next
- The search portion of a rule-based system
    - Takes advantage of heuristic information
    - Otherwise, the time to solve a problem could become prohibitively long
    - This problem is called the combinatorial explosion
- Expert-system shell provides customizable inference engine

2.2.4 Knowledge Base (rule base)

- Contains much of the problem solving knowledge
- Rules are of the form IF condition THEN action

- Condition portion of the rule is usually a fact - (If some particular fact is in the database then perform this action)

- Action portion of the rule can include

- Actions that affect the outside world (print a message on the terminal)

- Test another rule

- Add a new fact to the database (If it is raining then roads are wet).

- Rules can be specific, a priori rules (e.g., tax law . . . so much for each exemption) - represent laws and codified rules

- Rules can be heuristics (e.g. If the meal includes red meat then choose red wine). "rules of thumb" - represent conventional wisdom.

- Rules can be chained together (e.g. "If A then B" "If B then C" since A--->B--- >C so "If A then C").

- (If it is raining then roads are wet. If roads are wet then roads are slick.)

- Certainty factors represent the confidence one has that a fact is true or a rule is valid

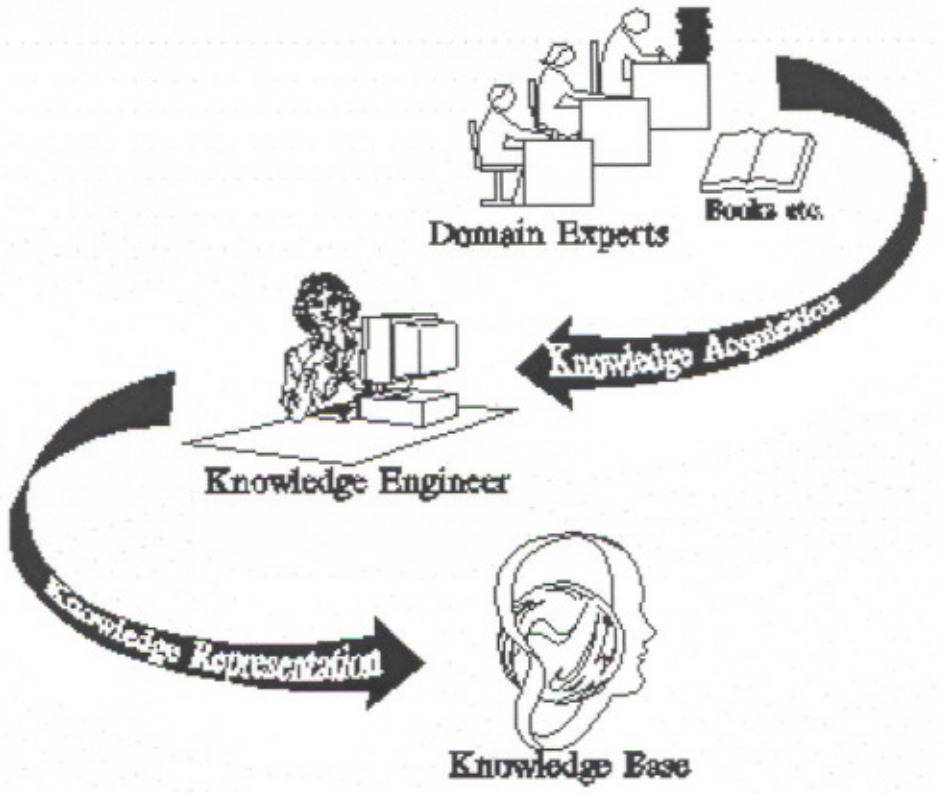

Fig 2.1 Role of Knowledge Engineer in an Expert System

2.3 Knowledge Engineering

Data is available in abundance. The data needs to be arranged and sorted in a manner that it is useful. Then some rules need to be formulated for interpreting that data. Data could be a record of last year's rainfall, a company's sales records, customer addresses or anything. Until it is formatted, it is not useful and cannot be called information. Data when organized and interpreted, gives us information, i.e. distilled data is information.

The discipline of building expert systems requires that we represent the available information in such a manner that it can be used/ accessed/ interpreted in a convenient manner. Knowledge is the intelligent application of data for obtaining desired results.

The various activities in Knowledge Engineering are:

2.3.1 Knowledge acquisition

The process of acquiring the knowledge from human experts or other sources (e.g. books, manuals) can involve developing knowledge to solve the problem.

2.3.2 Knowledge elicitation
Coaxing information out of human experts.

2.3.3 Knowledge representation
Method used to encode the knowledge for use by the expert system.

Common knowledge representation methods include rules, frames, and cases. Putting the knowledge into rules or cases or patterns is the knowledge representation process.

2.4 Types of Expert Systems

Two types of expert systems can be built with Prolog.

2.4.1 Production Systems
Production system is a type of expert system in which the knowledge is stored as rules and facts. Each rule is said to be a production rule. The rules express relationships between fact.

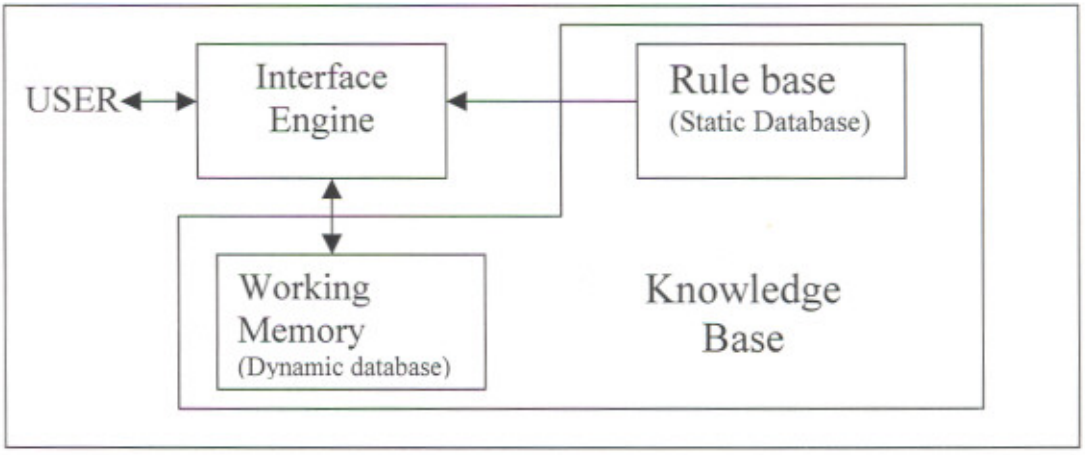

Fig 2.2: The Production System

A Production system has two primary components:

1. The Knowledge Base: It is the data or knowledge used to make decisions. The knowledge base contains the rules and facts about domains. It consists of two parts:

- Dynamic Database (Working Memory).
- Static Database (Rule Base).

The rule base consists of facts and rules that are compiled as part of the problem. These do not change during a particular consultation. The rule base corresponds to Prolog's static database.

The working memory consists of the facts relative to a particular consultation. The working memory corresponds to Prolog's dynamic database.

2. The Interface Engine: The interface engine has two functions: interface and control. Interface is the basic formal reasoning process. It involves matching and unification; it uses facts that are known to derive new facts by means of rules. Such interfaces operate by *modus ponens*.

The control function determines the order in which the rules are tested and what happens when a rule succeeds or fails. Prolog is its own interface engine. It performs unification, determines the order in which to scan the rules, and performs conflict resolution.

The static database contains two types of rules: knowledge base rules and control rules.

2.4.2 Frame-Based Systems

In this type of expert system a frame represents the object. The frame contains one or more filler slots, with each slot representing an attribute. Attribute values are stored in the slots. A slot can also contain limit values for another slot.

Frame-based systems are most applicable to biological classification system and similar types of systems, in which a static hierarchical classification is a part of the knowledge. Frame-based systems assume that the hierarchical relationship of the objects is relatively static. If the order is dynamic, using a frame-based system becomes difficult.

2.5 Building An Expert System

Expert systems are the end products of knowledge engineers. To build an expert system that solves problems in a given domain, a knowledge engineer starts by reading domain-related literature to become familiar with the issues and the terminology. With that as a foundation, the knowledge engineer then holds extensive interviews with one or more domain experts to "acquire" their knowledge. Finally, the knowledge engineer organizes the results of these interviews and translates them into software that a computer can use.

The interviews take the most time and effort of any of these stages. They often stall system development. For this reason, developers use the term knowledge acquisition bottleneck to characterize this phase.

2.6 Representing The Knowledge

The format that a knowledge engineer uses to capture the knowledge is called a knowledge representation. The most popular knowledge representation is the production rule (also called the if-then rule). Production rules are intended to reflect the "rules of thumb" that experts use in their day-to-day work. These rules of thumb are also referred to as

heuristics. A knowledge base that consists of rules is sometimes called a rule base.

2.7 Development Tools

One of the first expert system development tools was a by-product of one of the first expert systems. In the 1970s, Stanford researchers developed a system called MYCIN. MYCIN contains a multitude of rules (coded in the computer language LISP) that represent medical knowledge and enable the system to perform medical diagnoses.

MYCIN's developers reasoned that removing the medical diagnosis rules should not affect the workings of the system's inference engine and global database. With the medical knowledge gone (resulting in a system they named EMYCIN-"E" for "empty"), they also reasoned that they could insert knowledge from other domains into the knowledge base and thus build a fully functioning expert system. Because they were right on both counts, the expert system shell was born.

2.7.1 Expert Systems Shells

An expert system shell contains pre-coded expert system components (including either backward chaining, forward chaining, or both). The knowledge base is empty, its framework intact. All of this makes it unnecessary to rebuild the components for each new expert system. The knowledge engineer just adds the knowledge.

2.8 Advantages and disadvantages of expert systems

2.8.1 Advantages of expert systems

1. **Permanence** - Expert systems do not forget, but human experts may
2. **Reproducibility** - Many copies of an expert system can be made, but training new human experts is time-consuming and expensive.
3. **Competency** - If there is a maze of rules (e.g. tax and auditing), then the expert system can "unravel" the maze.
4. **Efficiency** - can increase throughput and decrease personnel costs.
5. **Inexpensive** - Although expert systems are expensive to build and maintain, they are inexpensive to operate. Development and maintenance costs can be spread over many users. The overall cost can be quite reasonable when compared to expensive and scarce human experts.
6. **Cost savings:** Wages - (elimination of a room full of clerks); Other costs - (minimize loan loss).

7. **Consistency** - With expert systems similar transactions handled in the same way. The system will make comparable recommendations for like situations. Humans are influenced by recency effects (most recent information having a disproportionate impact on judgment) primacy effects (early information dominates the judgment).

8. **Documentation** - An expert system can provide permanent documentation of the decision process.

9. **Completeness** - An expert system can review all the transactions, a human expert can only review a sample.

10. **Timeliness** - Fraud and/or errors can be prevented. Information is available sooner for decision making.

11. **Breadth** - The knowledge of multiple human experts can be combined to give a system more breadth that a single person is likely to achieve.

12. **Entry barriers** - Expert systems can help a firm create entry barriers for potential competitors.

13. **Differentiation** - In some cases, an expert system can differentiate a product or can be related to the focus of the firm (XCON).

2.8.2 Disadvantages of Expert Systems

1. **Common sense** - In addition to a great deal of technical knowledge, human experts have common sense. It is not yet known how to give expert systems common sense.

2. **Creativity** - Human experts can respond creatively to unusual situations, expert systems cannot.

3. **Learning** - Human experts automatically adapt to changing environments; expert systems must be explicitly updated. Case-based reasoning and neural networks are methods that can incorporate learning.

4. **Sensory Experience** - Human experts have available to them a wide range of sensory experience; expert systems are currently dependent on symbolic input.

5. **Degradation** - Expert systems are not good at recognizing when no answer exists or when the problem is outside their area of expertise.

2.9 CONCLUSION

Expert system stores the knowledge of one or more human experts, called domain experts, in a particular field. The system applies its stored knowledge to solve the problem. Knowledge-based expert systems use human knowledge to solve problems that normally would require human intelligence. Conventional computer programs perform tasks using conventional decision-making logic, containing little knowledge other than the basic algorithm for solving that specific problem and the necessary boundary conditions. This program knowledge is often embedded as part of the programming code, so that as the knowledge changes, the program has to be changed and then rebuilt.

Expert systems are the end products of knowledge engineers. To build an expert system that solves problems in a given domain, a knowledge engineer captures knowledge from a particular domain, organises and translates the knowledge into software that a computer can use.

Knowledge-based systems collect the small fragments of human know-how into a knowledge base which is used to reason through a problem, using the knowledge that is appropriate. A different problem, within the domain of the knowledge base, can be solved using the same program without reprogramming. Only the knowledge bases and inference engines that reside in our heads limit the possibilities. The next chapter describes the programming language Prolog that has been used to develope the expert system.

Chapter 3

Prolog

Introduction

PROLOG is a programming language based on ideas of logic programming. In fact, the word PROLOG means PROgramming in LOGic.

Prolog Programming in Logic combines the concepts of logical and algorithmic programming and is recognized not just as an important tool in AI Artificial Intelligence and expert systems but as a general purpose high-level programming language with some unique properties.

Prolog is unique in its ability to infer (derive by formal reasoning) facts and conclusions from other facts.

3.1 History

The language originates from work in the early 1970s by Robert A. Kowalski while at Edinburgh University and Alain Colmerauer at the University of Aix-Marseilles in France. Their efforts led in 1972 to the use of formal logic as the basis for a programming language. Kowalski's research provided the theoretical framework while Colmerauers gave rise to the programming language Prolog. Colmerauer and his team then built the frst interpreter and David Warren at the AI Department University of Edinburgh produced the first compiler.

3.2 The Language

A Prolog program is a set of procedures (the order is indifferent), each procedure consists of one or more clauses (the order of clauses is important). The crucial features of Prolog are unification and backtracking. Unification shows how two arbitrary structures can be made equal, and Prolog processors employ a search strategy, which tries to find a solution to a problem by backtracking to other paths if any one particular search comes to a dead end.

Prolog is a meta-programming language. It is good for problems in which logic is intimately involved, or whose solutions have a logical characterization. Like other interactive, symbolic languages, Prolog is also good for rapid prototyping.

3.2.1 Top down computation

Top down computation, typically used in Prolog, starts with the original problem and it decomposes this problem to simpler and simpler problems till the trivial problem, i.e., the fact in Prolog database, is reached. Then, the solution of larger problems is composed of the solutions of simpler problems etc. till the solution of the original problem is obtained. Where as the bottom up computation starts with know facts and extends the set of know trues using rules, i.e. it derives new facts from old facts and rules.

3.2.2 Prolog is Object Oriented

Prolog is a pure Object Oriented Language. It uses no procedures and essentially no programs.

- Prolog uses only data about objects and their relationships.
- Prolog also emphasizes symbolic processing.

3.2.3 Prolog is Procedure Less

With Prolog, the user defines a goal (a problem or objective), and the computer must find both the procedure and the solution.

A Prolog program is a collection of data or facts and the relationships among these facts. In other words the program is a database.

3.3 Comparison of Prolog with Procedural Languages

Procedural Languages **(BASIC, COBOL, Pascal, C)**	**Object Oriented Languages** **(Prolog, LISP)**
Uses previously defined procedures to solve problems.	Uses heuristics to solve problems.
Most efficient at numerical processing.	Most efficient at formal reasoning.
Systems created and maintained by programmers.	Systems created and maintained by knowledge engineers.
Use structured programming.	Interactive and cyclic development.

Fig 3.1: Comparison of Procedural and Non-Procedural Languages

Procedural languages use algorithms. Algorithms are objective procedures, which when followed, guarantee a solution.

Prolog use heuristics. Heuristics are rules of thumb that are useful in reaching a goal. But they do not guarantee a solution, nor do they always point to the most efficient solution. They are useful, when there no algorithm exists.

3.4 Structure of a Prolog program

In designing an expert system, the expert abstracts reality to create a mental picture that has both objective and subjective aspects.

The knowledge engineer takes the expert's mental abstraction and builds a model called a knowledge representation, This knowledge representation is then converted into a program in Prolog, that behaves, in theory at least, as the real world does.

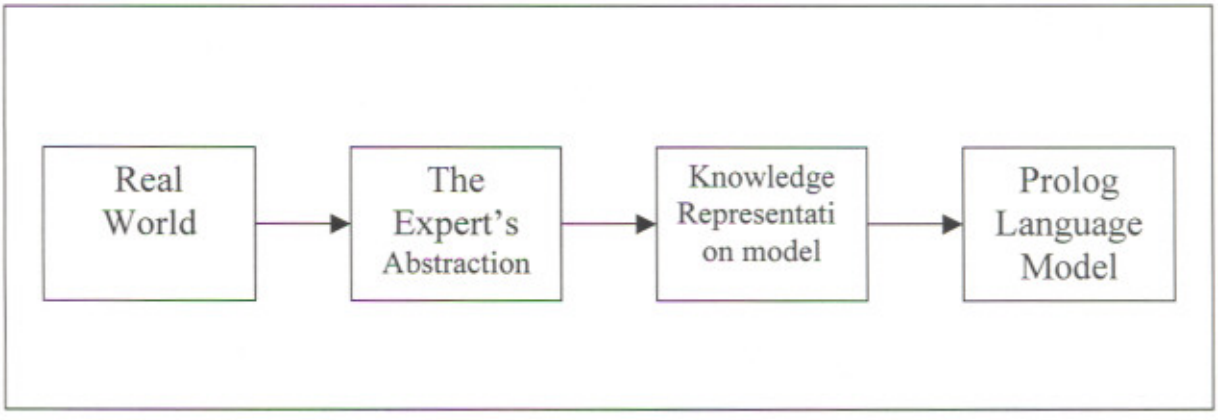

Fig 3.1 Modeling the real world using Prolog.

A Prolog program consists of two or more sections.

- Clauses
- Predicates

Besides these a Prolog program can also have the following section: "Domains"

3.4.1 Clauses

The main body of the program, the clauses section, contain the clauses and consists of facts and rules.

3.4.2 Predicates

The relations used in the clauses section are defined in the predicates section. Each clause must have a corresponding predicate definition in the predicates section.

3.4.3 Domains

It defines the type of each object. Many implementation of Prolog allow one to skip this section, but it is useful as it allows some control over the objects types. Also the program becomes easier to understand. Six basic object types are available:

- Char
- Integer

- Real
- String
- Symbol
- File

3.5 Prolog Objects

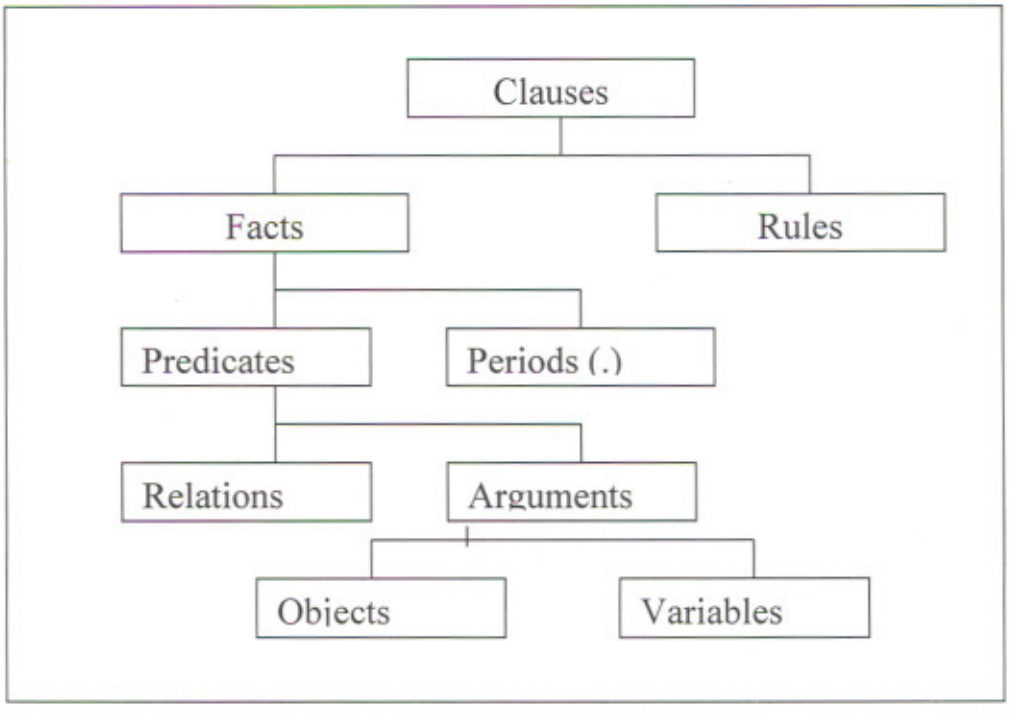

Fig 3.2 Relationship of Clauses, Predicates, relations and objects.

The main part of a Prolog program consists of a collection of knowledge about a specific subject. This collection is called a database, and this database is expressed in facts and rules.

Prolog permits to describe facts as symbolic relationships.

The factual expressions (facts) in Prolog are called *clauses*.

A *relation* is a name that defines the way, in which a collection of objects (or objects and variables referring to objects) belongs together.

A *predicate* expresses a property or a relationship.

A *rule* is an expression that indicates that the truth of a particular fact depends upon one or more other facts. Rules permit Prolog to infer new facts from existing facts.

3.6 Prolog Execution Rules

- Prolog executes using a matching process.
- When the original goal is specified, Prolog tries to find a fact or the head (conclusion) of a rule that matches this goal.
- If a fact is found, the goal succeeds immediately.
- If a rule is found, Prolog then tries to prove the head of the rule by using the antecedent (the body of premises) as a new compound goal and proving each premise of the antecedent. If any premise of the antecedent fails, Prolog backtracks and tries to solve the predicating premises with other bindings. If Prolog is not successful, it tries to find another fact or rule that matches the original goal. If all premises succeed, the original goal succeeds.
- Prolog continues to execute until all possible solutions for the goal are tested.
- Execution is from top to bottom and from left to right. Prolog uses depth first testing; i.e. testing of any particular rule will proceed as far as it can until all subgoals fail, and then the next rule is tested to see if it unifies with the specified goal.

3.7 The Standards in Prolog

Until now there was no standard in Prolog. Recently the ISO (The International Organization for Standardization) and IEC (The International Electrotechnical Commission) formulated the first International Standard for Prolog on July, 1999. This standard "**International Standard ISO/IEC 13211**" was prepared by "Joint Technical Committee ISO/IEC JTC 1, Information technology Subcommittee SC 22, Programming languages, their environments and system software interfaces".

Before that, the closest thing to an accepted prolog standard was the Prolog described in Clocksin and Mellish's book Programming in Prolog (also known as C&M Prolog).

3.9 Conclusion

A Prolog program is a collection of data or facts and the relationships among these facts. In other words the program is a database. And as a knowledge engineer it is our task to organize and construct that database in an efficient and effective manner.

Prolog permits to describe facts as symbolic relationships. Also Prolog is a pure Object Oriented Language, besides providing its own inference engine. This makes the task of creating Expert Systems a little easier, for a knowledge engineer.

Chapter 4

Expert System for Automobile Problem Diagnostics & Maintenance

Introduction

The expert system for "**Automobile Problem Diagnostics and Maintenance**" facilitates a person with moderate knowledge about automobiles to diagnose and rectify problems related to automobiles. It also assists a real expert to quickly diagnose problems related with automobiles.

The knowledge base (database) of the expert system is extensible. The knowledge base can be updated to include information about unknown and new problems.

The system is flexible enough to permit the use of a different knowledge base to diagnose similar problems in different domains with very little changes.

4.1 Nature of the Problem

Finding and diagnosing problems in Automobiles is not an easy task. Everyday new models are being released in the market. It is not an easy task to keep track of each and every model and the problems occurring in each of them for a human being even if he is an expert in the field. Thus use of a computer system to diagnose and probably suggest solutions is justified.

Computer systems are easy to extend, maintain, and copy to other installations. Though they do require extensive efforts to develop, but the work done is worth the effort.

Though the problem diagnostics for Automobiles require the user to be near the automobile and be a little knowledgeable in Automobiles (i.e. they should at least be familiar with various parts of the automobile's system). But still this system is a great utility for those who have to diagnose problems in Automobiles, as they don't have to tread into a blind ally if they don't know much about automobiles.

All the problems cannot be diagnosed by just asking yes/no type questions, but by careful mapping of the knowledge a knowledge engineer can considerably reduce the overhead of diagnosing problems in automobiles.

4.1.1 Expected Audience of the System

Those people, who have to deal with an automobile everyday, or frequently, like a mechanic, automobile owner, engineer, service personnel, workshops etc.

- Service Engineers
- Mechanics
- Automobile Workshops
- Quality Assurance Departments of Automobile Firms
- Training facilities for Automobile Engineers & mechanics.
- Webmasters of Automobile Websites
- Users of Automobiles

4.1.1 What the system requires/ expects from its user

The user of the system should at least be familiar with various parts of the automobile. User need not be an expert, but that is an added advantage. The program would ask the user to check for various problems and would conclude its result from this questioning.

4.2 Considerations in The Design of Expert System

Artificial Intelligence leaves behind the comfortable algorithmic certainty of conventional programming. Instead of simply computing our answers, problems are now so complex that we must search for solutions as humans do. Though there is no guarantee that the search will yield an answer or if it does, that the answer will be optimal.

The biggest problem in the design of an expert system is abstracting the knowledge of the expert and putting it into an objective form for the database.

A human automobile expert normally approaches a problem atleast partially subjectively. The computer, in contrast, can work only with objective representations of knowledge about automobiles. Until formal rules and facts exist, the expert system in the field of automobiles cannot be designed.

Expert systems are designed by *knowledge engineers*, It is the knowledge engineer that must observe, talk to, and work with the human expert to determine how to express the reasoning process of the human expert in an objective form.

4.3 Why Use Prolog ?

Conventional programming languages, such as FORTRAN and C/C++, are designed and optimized for the procedural manipulation of data (such as numbers and arrays). Humans, however, often solve complex problems using very abstract, symbolic approaches which are not well suited for implementation in conventional languages. Although abstract information can be modeled in these languages, considerable programming effort is required to transform the information to a format usable with procedural programming paradigms. This can often distract an expert system designer from the original goal.

One of the results of research in the area of artificial intelligence has been the development of techniques, which allow the modeling of information at higher levels of abstraction. These techniques are embodied in languages or tools which allow programs to be built, that closely resemble human logic in their implementation and are therefore easier to develop and maintain.

Prolog is one such tool, it uses heuristics to solve problems, besides compilers for Prolog are easily available. It allows representing knowledge in an abstract form, which is closer to the human thinking procedure. The inference engine is built into the Prolog compilers and interpreters.

Prolog is a mature language, the learning curve is not so steep as compared to other such languages like LISP. The syntax is simple and easy to understand. That's why Prolog has been selected for developing the expert system.

4.4 Rule Based Expert System

This expert system is rule-based. Rule-based programming is one of the most widely used techniques for developing expert systems. In this programming paradigm, rules are used to represent heuristics, or "rules of thumb," which specify a set of actions to be performed for a given situation.

A rule is composed of an **if** portion and a **then** portion. The **if** portion of a rule is a series of patterns which specify the facts (or data) which cause the rule to be applicable. The process of matching facts to patterns is called pattern matching. The expert system tool provides a mechanism, called the inference engine, which automatically matches facts against patterns and determines which rules are applicable. The **if** portion of a rule can actually be thought of as the whenever portion of a rule since pattern matching always occurs whenever changes are made to facts. The **then** portion of a rule is the set of actions to be executed when the rule is applicable. The actions of applicable rules are executed when the inference engine is

instructed to begin execution. Example of such an **if then** rule is as follows:

```
IF engine_getting_petrol
AND engine_turns_over
THEN problem_with_spark_plugs
```

The inference engine selects a rule and then the actions of the selected rules are executed (which may affect the list of applicable rules by adding or removing facts). The inference engine then selects another rule and executes its actions. This process continues until no applicable rules remain.

4.5 Domain and Goals of the System

During the last century, civilization has undergone a knowledge explosion, and it has become impractical if not impossible for an individual to gain ever area of human endeavor. We are a species of specialists. Similarly, intelligent computer systems must also specialize. Instead of trying to design a computer capable of expertise in several fields, the focus must be on a single, carefully chosen, clearly defined subject.

The objective of the expert system for "**Automobile Problem Diagnostics and Maintenance**" is to identify the problem in an automobile from the information provided by the user of the system.

Though the domain is limited to four wheeled-automobiles, but it can be extended to other automobiles as well. Some problems in the system are quite general and are applicable to other automobiles as well.

4.5.1 Features of the System

- The system is capable of diagnosing common problems in Automobiles.
- It can suggest possible rectification for the problem it has identified.
- An Engineer, Mechanic or Individual can easily correct problems using this system.
- An expert Automobile Engineer can easily extend the knowledge base of the system.
- If the system is unable to identify the problem it warns user of this and prompts the user to update its database.

4.6 Structure of the system

Rule-based systems can be either goal driven using backward chaining to test whether some hypothesis is true, or data driven, using forward chaining to draw new conclusions from existing data. Expert system for "**Automobile Problem Diagnostics and Maintenance**" may use either or both strategies, but the most common is probably the goal driven backward chaining strategy.

One reason for this is that normally an expert system will have to collect information about the problem from the user by asking them questions - by using a goal driven strategy we can just ask questions that are relevant to a hypothesized solution.

The expert system will consider each hypothesized solution and try to prove whether or not it might be the case. Sometimes it won't be able to prove or disprove something from the data initially supplied by the user, so it will ask the user some questions. Using any initial data plus answers to these questions it should be able to conclude which of the possible solutions to the problem is the right one.

4.7 Building the System

The reasoning process of this expert system "**Automobile Problem Diagnostics and Maintenance**" is a search through the problem space. On path to the eventual goal are sub-goals. This system starts with one goal and tries to prove it by proving sub-goals. If this proof fails, the system moves to next goal and tries to prove it.

The problem space consists of nodes and links. Each node represents both a symptom and a subgoal. A subgoal might represent a symptom "Engine Fails to Start". The expert system verifies that this node has been reached in the problem space by asking whether "the engine starts or not". The same node represents a subgoal of all problems for which "Engine Fails to Start" is a symptom. By asking questions and firing rules, the system moves from one node to the next.

For diagnostics systems like the "**Automobile Problem Diagnostics and Maintenance**" the higher-level rules take the form

```
hypothesis( Z ):-
    symptom( X1 ),
    symptom( X2 ),
    .
    .
    .
    symptom( Xn ).
```

Here the system would proceed as follows to prove the hypothesis 'Z':

- First it would try to prove 'X1'.
- If 'X1' is true, than it would try to prove 'X2'.
- If 'X2' is true, than it would try to prove 'X3'.
- And so on, thus if 'Xn' is true, than hypothesis is proved, and a solution has been located.
- If anyone of these fails, then the hypothesis fails.

The knowledge base facts can be as follows:

```
Problem: Low oil pressure
    Worn engine bearings
    Engine over heating
    Oil dilution or foaming
    Lubrication-system defects
...
...
...

Etc.
```

This list can also be used to test the system manually.

The above can be represented as Prolog predicates as follows.

```
hypothesis(low_oil_pressure ):-
    symptom(worn_engine_bearings ),
    symptom(engine_over_heating ),
    symptom(oil_dilution_or_foaming ),
    symptom(lubrication_system_defects ).
```

4.8 Working of the System (Sample Run)

This is much better explained through a simple example. Suppose that we have the following rules for the system:

```
IF engine_getting_petrol
AND engine_turns_over
THEN problem_with_spark_plugs

IF NOT engine_turns_over
AND NOT lights_come_on
THEN problem_with_battery
```

```
IF NOT engine_turns_over
AND lights_come_on
THEN problem_with_starter

IF petrol_in_fuel_tank
THEN engine_getting_petrol
```

These rules can be represented in Prolog as follows

```
hypothesis( problem_with_spark_plugs ):-
    Symptom( engine_getting_petrol ),
    Symptom( engine_turns_over ).

hypothesis(problem_with_battery ):-
    Symptom( engine_turns_over ),
    Not( Symptom( lights_come_on ).

hypothesis( problem_with_starter ):-
    Symptom( engine_turns_over ),
    Symptom( lights_come_on ).

hypothesis(engine_getting_petrol ):-
    Symptom( petrol_in_fuel_tank ).
```

Our problem is to work out what's wrong with our car given some observable symptoms. There are three possible problems with the car: problem_with_spark_plugs, problem_with_battery, problem_with_starter. We'll assume that we have been provided with no initial facts about the observable symptoms.

In the goal-directed system we would try to prove each hypothesised problem (with the automobile) in turn. First the system would try to prove "problem_with_spark_plugs"'. Rule 1 is potentially useful, so the system would set the new goals of proving "engine_getting_petrol"' and "engine_turns_over"'. Trying to prove the first of these, rule 4 can be used, with new goal of proving "petrol_in_fuel_tank"' There are no rules which conclude this (and the system doesn't already know the answer), so the system will ask the user:

Is it true that there's petrol in the fuel tank?

Let's say that the answer is yes. This answer would be recorded, so that the user doesn't get asked the same question again. Anyway, the system now has proved that the engine is getting petrol, so now wants to find out if the engine turns over. As the system doesn't yet know whether this is the case, and as there are no rules, which conclude this, the user will be asked:

Is it true that the engine turns over?

Lets say this time the answer is no. There are no other rules, which can be used to prove "problem_with_spark_plugs'" so the system will conclude that this is not the solution to the problem, and will consider the next hypothesis: problem_with_battery. It is true that the engine does not turn over (the user has just said that), so all it has to prove is that the lights don't come one. It will ask the user

Is it true that the lights come on?

Suppose the answer is no. It has now proved that the problem is with the battery. System might stop here, but usually there might be more than one solution, (e.g., more than one fault with the car), or it will be uncertain which of various solutions is the right one. So usually all hypotheses are considered. It will try to prove "problem_with_starter", but given the existing data (the lights come on) the proof will fail, so the system will conclude that the problem is with the battery. A complete interaction with the system might be:

System: Is it true that there's petrol in the fuel tank?
User: Yes.
System: Is it true that the engine turns over?
User: No.
System Is it true that the lights come on?
User: No.
System: I conclude that there is a problem with battery. Recharge or Replace the battery.

Here is another example with following rules:

IF engine_over_heats
AND engine_lacks_power
AND choke_struck_partially_open
THEN repair_free_valve

IF engine_over_heats
AND engine_lacks_coolent
THEN replace_coolent

IF engine_over_heats
AND loose_or_broken_fan_belt
THEN replace_fan_belt

IF engine_over_heats
AND ignition_timing_late
THEN adjust_ignition_timing

In all the above rules, the initial complaint is common, i.e. engine over heats. But engine can overheat due to many other problems with the

automobile also. The inference engine of the system select only that solution for which all the symptoms prove to be true. Rejecting all those for which any of the symptoms is not true.

4.9 Conclusion

Expert System for "**Automobile Problem Diagnostics and Maintenance**" is rule-based system and rule-based programming is one of the most extensively used techniques for developing expert systems. In this programming paradigm, rules are used to represent heuristics, or "rules of thumb," which specify a set of actions to be performed for a given situation.

The rules of thumb have been defined from knowledge about automobiles and their problems. The rules of thumb were gathered from books, magazines, and articles, and were mapped into the knowledge base of the system. Different rules have been defined for the problems occurring in an automobile. The system tries to prove a goal (where the goal is diagnostic and ultimately a solution for the problem) by inquiring the user about different symptoms for different problems. Heuristic techniques have been incorporated to minimize the number of questions asked. The system remembers the answers to those questions, which have been previously answered, and tries to infer a solution from the information available and gained from the user. If a solution is found, it is displayed, else the user, if is an expert, is prompted to update the knowledge base for that specific problem.

Expert System for "**Automobile Problem Diagnostics and Maintenance**" tries to solve problems in automobiles, as a human expert for automobiles would do. It is quite easy to use, with an extensible knowledge base. It can act as a utility for a human expert as well.

Chapter 5

Conclusions & Further Scope

Conclusions

This thesis analyses Expert Systems and their application in the field of automobile problem diagnosis and maintenance. The work presented here can be summarized as under:

- The knowledge from the domain of human automobile experts has been mapped into the expert system as **if-then** rules using Prolog predicates. The knowledge base has been acquired from books, magazines, Internet, human experts. The knowledge base of the system can be extended to include information about new models or manufacturers with ease. The inference engine of Prolog searches through these rules of thumb, using backward chaining, to prove a goal, which satisfies a particular problem.

- Knowledge representation has been done as predicates, which can be extended to include new rules and facts. The Inference engine of Prolog explores the hypothesis and is able to diagnose a problem, with sufficiently accurate and much better results than a human expert in similar circumstances. The expert system does not forget, but human experts may, thus the rules and predicates once defined remain in the memory of the computer. Many copies of an expert system can be made, but training new human Automobile Experts is time-consuming and expensive. This increases throughput and decreases personnel costs, and is economical to operate. The overall cost can be quite reasonable when compared to expensive and scarce human experts. Humans are influenced by recency effects (most recent information having a disproportionate impact on judgment) primacy effects (early information dominates the judgment). But the performance of the expert system developed does not vary. The knowledge of multiple human experts can be combined to give this expert system more breadth than a single person is likely to achieve.

- The user interface of the system has been developed in the C programming language, which acts as a shell to the actual expert system, which has been developed in Prolog. This system is user friendly, easy to use and requires the user to answer Yes/No for each question asked. Answering Yes or No is easy as compared to detailed descriptive explanation, which in case of human experts is required. The user need not remember each and every detail about any

automobile system. It is a utility for human experts also. This expert system can also be used to provide training to other human experts.

5.1 Further Scope

- This system can be modified to generate HTML code directly for responding to user queries on the Internet, using a web browser. Thus Automobile Manufacturers can use this system to diagnose and sort out problems online, for their automobiles, without using a human expert. A JAVA based Prolog interpreter "Jinni 2.27" is also available at http://www.binnetcorp.com/Jinni. It is a Multi-threaded, Java based Prolog interpreter with built-in networking support and many other features.

- The knowledge base can be extended to incorporate the automobiles of a specific manufacturers or model.

- This system can be ported to many different platforms, which have an ISO standard Prolog compiler or interpreter available.

- This system can be used as a training tool for human experts, they can query the system for any problems.

This system can be extended to incorporate many new techniques. Only the imagination and efforts are the limiting factor.

References

[1] Rich, Elaine and Knight, Kevin. Artificial Intelligence Second Edition. 1991, McGraw-Hill.

[2] Waterman, Donald A. A Guide to Expert Systems, 1986, Reading, Addison-Wesley.

[3] Townsend, Carl. Introduction to Turbo Prolog 1st Edition. 1988, SYBEX Inc.

[4] Patterson, Dan W. Introduction to Artificial Intelligence and Expert Systems 6th Reprint. 1999, PHI.

[5] Liebowitz, Jay, Introduction to Expert Systems, 1988, Santa Cruz, Mitchell Publishing, Inc.

[6] Brule, James F. Artificial Intelligence: Theory, Logic and Application, 1986, Blue Ridge Summit, TAB Books.

[7] Michaelsen, Robert H.; Michie, Donald and Boulanger, Albert. "The Technology of Expert Systems" Byte; April 1985; 10(4): pp. 303-312.

[8] Artificial Intelligence Programming in Prolog [Alison Cawsey] - http://www.cee.hw.ac.uk/~alison/ai3notes/all.html

[9] On-Line Guide to Prolog Programming [Roman Bartak] - http://kti.ms.mff.cuni.cz/~bartak/prolog/index.html

[10] Logic Programming home page http://www.logic-programming.org

[11] ISO Prolog Standards - http://www.logic-programming.org/prolog_std.html

[12] Logic-Programming-Archive-[J.Bowen]- http://archive.comlab.ox.ac.uk/logic-prog.html

[13] Databases and Artificial Intelligence - http://www.cee.hw.ac.uk/~alison/ai3notes/all.html

[14] An Overview of Expert Systems - http://www.friends.edu/gradCOB/EMBA/MBA652/expert/index.htm

[15] INTRODUCTION TO ARTIFICIAL INTELLIGENCE AND EXPERT SYSTEMS - http://www.bus.orst.edu/faculty/brownc/es_tutor/es_tutor.htm

[16] The Genetic Programming Notebook - http://www.geneticprogramming.com/jjf/

[17] Evolutionary Computation and Artificial Life Page - http://www.ai.mit.edu/people/unamay/EC.html

[18] Informing Science - http://inform.nu.

[19] Expert System FAQ - http://www.cs.cmu.edu/Web/Groups/AI/html/faqs/ai/expert/part1/faq.html

[20] The AI Page [Aaron Sloman] - http://www.cs.bham.ac.uk/~axs/misc/aiforschools.html

[21] CLIPS Home Page http://www.jsc.nasa.gov/~clips/CLIPS.html

[22] IEEE Expert http://www.computer.org/intelligent/

[23] Neural Technologies http://www.neuralt.com/neural_technologies.htm

[24] FuzzyTECH http://www.fuzzytech.com/

~0~