Monograph:
Wheatley, M.D. (1984) Expert Systems in Transport – Part 1: An Introduction to Expert Systems. Working Paper. Institute of Transport Studies, University of Leeds , Leeds, UK.

Working Paper 178

Working Paper 178

July 1984

EXPERT SYSTEMS IN TRANSPORT

Part 1: An introduction to expert systems

M.D. Wheatley

This work was sponsored by the University of Leeds.

ABSTRACT

WHEATLEY, M.D. (1984) Expert systems in transport. Part 1: An introduction to expert systems. Working Paper 178, Institute for Transport Studies, University of Leeds, Leeds.

This report describes what expert systems are. It has been written to introduce the concepts involved, and to some extent the techniques, for those who have no previous acquaintance with such systems, but who are concerned with establishing quite what such systems might have to offer, and what is involved in developing them, in their field of application. Ways in which knowledge and uncertainty are represented in expert systems are described, and illustrated by reference to some existing expert systems. The report stems from a project designed to assess the potential for establishing expert systems in the transport field, and discusses the types of expert system package which might be useful in the remaining stages of the project. It includes a useful set of references and a glossary.A preliminary assessment of potential transport applications and the implications for the remainder of the project are described in a companion report, ITS Technical Note 145.

KEYWORDS: EXPERT SYSTEMS; ARTIFICIAL INTELLIGENCE; RULE-BASED SYSTEMS; KNOWLEDGE; TRAFFIC; TRANSPORT;

CONTENTS

Contents (continued)

EXPERT SYSTEMS IN TRANSPORT

Part 1: An introduction to expert systems

1. INTRODUCTION

1.1 Project Background

This project is an investigation into how expert systems, and associated computing techniques, can be applied to the field of transport. The project was conceived following the stimulus given by the Alvey Committee's report on Advanced Information Technology (Alvey, 1982), which (successfully) proposed that Government funds be put into the development of fundamental research into Intelligent Knowledge-Based Systems (IKBS); and which also called for the exploitation of the technology already available in the artificial intelligence (AI) field. In the transport sector, it was particularly timely that ITS should take the initiative in this area: a Science and Engineering Research Council (SERC) Visiting Fellow, Dr M.R. Wigan (Australian Road Research Board), was identifying potential research areas in information technology and transport (Wigan, 1983, Kirby 1984), and paid special attention to work with rule-based systems. (Wigan 1984); and the Joint Committee of SERC and the Economic and Social Research Council (ESRC) had just set up a special programme to develop projects that applied information technology to transport. The project was funded as a 2-year University of Leeds research post, from November 1983.

Already it can be seen that four terms (expert systems; IKBS; artificial intelligence; rule-based systems) have been (purposely) used to represent the type of computing system in which this project is interested, and it is symptomatic of the whole subject area that these terms are often used inter-changeably and without explanation, even though they have different shades of meaning. It is hoped that this report will clarify the terminology, and present the concepts and techniques involved in a clear and concise manner.

1.2 Project Aims

The aims of the project are not defined in terms of producing a stated end product, but in terms of developing a general understanding of expert systems within the Institute, and of demonstrating to transport professionals the power and flexibility of expert systems as a problem solving tool. In the course of this, it is intended to develop a demonstration expert system which will go some way to fulfilling both the above major aims, as well as providing the Institute with a new problem solving tool for general use. In the course of this, an assessment is being made of a range of potential transport-related problems which can be solved by the application of an expert system; the assessment prepared so far (February 1984) is presented in a separate report (ITS Technical Note 145).

1.3 Project Status

It is a characteristic of this subject that expertise in expert systems is not readily available, and people with a thorough knowledge of the subject and the ability to communicate it to others are rare indeed. The author not being one of these rare creatures, the initial part of the project has been almost completely concerned with overcoming the not inconsiderable difficulties involved in understanding what expert systems are and, to some extent, how they work.

This report presents the results from this initial phase of the project. Much work has been done in trying to become familiar with expert systems, and to communicate this familiarity to Institute staff; it is thought that some success has been achieved in the latter by means of a series of colloquia held within the Institute, and it is hoped that the interest generated will continue to develop. This report and its companion Technical Note will help in identifying the potential that expert systems have in transport.

The practical side of the work has not yet started; it was considered to be important, and indeed was part of the project aims, that familiarisation with techniques should have first priority. It is very tempting to say that proper familiarisation can only be brought about with 'hands-on' experience, but in the case of expert systems the availability of software and appropriate computing resources has been a limiting factor.

However, the project is now ready to move into the construction of a demonstration expert system, using tools identified as most appropriate in this initial phase.

1.4 Report Structure

Chapter 1 states details of the project - its background, aims and status.

Chapter 2 introduces the subjects of artificial intelligence and expert sytems, explaining their aims and outlining their pitfalls for the unwary novice researcher.

Chapter 3 looks in more detail at the construction of expert systems.

Chapter 4 examines a selection of current expert systems and expert system shells.

A separate Technical Note contains a chapter on the relationship of expert systems to the field of transport generally, examining the pros and cons of possible application areas, and reviews the specific position of ITS with respect to expert systems in general, and the remainder of this project in particular.

A glossary of technical terms is provided at Section 7.

2. THE CONCEPTS BEHIND EXPERT SYSTEMS

2.1 Computers in General

The kind of thing to which computer technology has traditionally been applied is the carrying out of repetitious physical tasks, such as controlling tools or solving numerical-based problems, such as statistical tests. It has always been considered that computer technology would never be able to rise from this somewhat mundane level to the level of making decisions or interpreting information on the basis of weighing evidence and arriving at a conclusion. This has always been something that only a human being can do, mainly for the somewhat paradoxical reason that human beings do not fully understand the processes they go through when evaluating information, and so could not attempt to program a computer to perform the same processes.

Computers are able to do things quickly - some of the larger mainframe computers can carry out millions of instructions per second - and so they are very useful in doing repetitive, boring things, leaving human beings free to do the creative, intuitive things that computers cannot do. However, some computing scientists, together with philosophers, psychologists and educationalists, are concerned with trying to give computers the power and ability to carry out these very tasks that have so far been totally the perogative of human beings. How is this being done? The answer is simple, the explanations are not!

There is a subject, branch of computing science, branch of psychology, call it what you will, known as 'Artificial Intelligence (AI) which, as it's name implies, is concerned with endowing machinery with that essentially human quality - intelligence. (There is much literature available concerning the high-flown intricacies of implementing this, but very little on the basics.) A branch of AI is the field known as 'Expert Systems', which concentrates on building computer systems which appear to have intelligence when used in very specific applications.

For those with some knowledge of computing, the development of expert systems is an important and interesting step in the development of software techniques (i.e. the design of programs to tell the computer what to do). For those with no knowledge of computing, then expert systems can represent a frightening step nearer Computers In Control. Let us look at how expert systems developed, and try to destroy this latter fear in the process.

2.2 A Step Back

It will no doubt already have occured to you that some strange ideas have been put forward in the above section, and not only that, but that they have been glossed over without explanation or definition. After all, how can a machine be given intelligence? What is intelligence? Wouldn't an intelligent computer be a contradiction in terms? The answer to all these questions, and

probably to all the others relating to intelligence is 'I do not know'. No psychologist will give a definition of 'intelligence', or even of 'thought'; so we have to proceed on the understanding that we all know what we mean by intelligence, but that we can't quite put it into words. The concept of an intelligent computer comes clearly to mind - you speak to it, it answers back (Eddie the Shipboard Computer in 'The Hitchhikers Guide to the Galaxy' [Adams, 1977] is my model - no doubt you can think of others!).

As this discussion progresses, a certain amount of anthropomorphism will inevitably creep in, let us state here and now that it is just a convenient way of referring to the concepts involved. The whole argument relating to intelligence and computers has yet to be resolved, but it does not prevent us looking at expert systems in some detail, and AI very briefly.

2.2.1 Getting into the subject. One of the most fascinating aspects of expert systems as a subject is that it is extremely new. Techniques tend to be developed, and then the theoretical justification for developing them is (sometimes) expounded. Often, practical implementation outruns theory, and almost always outruns documentation. Documentation is, thus, very hard to come by, particularly so if one is looking for introductory texts; it can also be said that many researchers already well ensconced in expert systems (not that there are that many of them), never descend to the layman's level, and will have totally confused anyone not conversant with 'rule-based sytems', 'frame-based systems', 'knowledge representation', 'meta-knowledge', etc. within 2 minutes conversation.

The great problem therefore, if one is trying to use or develop expert systems for a particular application, and if one does not really know a lot about them, is that of obtaining information about expert systems that actually makes sense. Fortunately, new articles and books are appearing all the time, as the circle of those wanting to know widens; some are listed in section 6.1. That by Feigenbaum and McCorduck (1984), is a good starting point; but one should be warned that some books, such as Naylor (1984) can oversimplify so much as to make it appear that one can do it all in BASIC! And that is certainly not the case for real applications...

We have already seen that AI is concerned with trying to give machines intelligence, and that expert systems can be regarded as a sort of practical off-shoot from AI. It is interesting to look at how these subjects came into being, and the type of things they have achieved so far. Looking at some AI systems, even if it is only reading the title of various articles, is actually very helpful in trying to understand what types of task the computer is being told to perform.

2.3 AI

The best way to explain AI is to quote unashamedly from one of the leading researchers in the field, E.A. Feigenbaum: "The

potential uses of computers by people to accomplish tasks can be 'one-dimensionalised' into a spectrum representing the nature of the instruction that must be given the computer to do its job. Call it the WHAT-to-HOW spectrum. At one extreme of the spectrum, the user supplies his intelligence to instruct the machine with precision exactly HOW to do his job, step-by-step. Progress in Computer Science can be seen as steps away from the extreme 'HOW' point on the spectrum: the familiar panoply of assembly languages, subroutine libraries, compilers, extensible languages, etc. At the other extreme of the spectrum is the user with his real problem (WHAT he wishes the computer, as his instrument, to do for him). He aspires to communicate WHAT he wants done in a language that is comfortable to him (perhaps English); via communication modes that are convenient for him (including perhaps, speech or pictures); with some generality, some vagueness, imprecision, even error; without having to lay out in detail all necessary subgoals for adequate performance - with reasonable assurance that he is addressing an intelligent agent that is using knowledge of his world to understand his intent, to fill in his vagueness, to make specific his abstractions, to correct his errors, to discover appropriate subgoals, and ultimately to translate WHAT he really wants done into processing steps that define HOW it shall be done by a real computer. The research activity aimed at creating computer programs that act as 'intelligent agents' near the WHAT end of the WHAT-to-HOW spectrum can be viewed as the long-range goal of AI research". (Feigenbaum, 1974).

It is said (personal communication) that AI came into being via robotics - researchers wanted to be able to give robots the ability to learn from experience, and to be able to react to things happening in the 'outside world' without necessarily having been programmed. In other words, to be given a set of rules describing proceedural behaviour given a specific set of circumstances; a set of facts describing possible sets of circumstances; and a mechanism for matching 'outside world' circumstances to the set of facts and thus deciding the appropriate behaviour by implementing the appropriate rule. Implicit within this desire is the ability to infer new rules and facts from the explicitly stated rules and facts, and thus to add to the information available to the robot. The set of rules and facts is known in the jargon as the 'knowledge base'; whilst the mechanism for inferring new rules and facts (which would then be added to the knowledge base), and matching observed facts to internally held facts, is known as the 'inference engine'.

It can be seen that Feigenbaum's statement above is not so far removed from the roboticists' aims - a computer needs 'knowledge' of a subject in order to be able to act intelligently when applied to that subject; and it needs to know how to use that knowledge in a specific situation.

So, with apologies for the anthropomorphism, AI is trying to give computers the ability to possess knowledge of a particular subject, in such a way that that knowledge can be used and added

to in a flexible way by both the user and the computer itself. Ideally implemented, it would make the computer 'transparent' to the user, i.e. make the user unaware that he is using a computer.

Before confusing everyone too much, and without even considering the philosophical arguments about whether such aims are possible (they have to be - they have to some extent already been achieved) - we shall move on from AI aims to look specifically at examples of AI systems. However, it is worth mentioning one very common method of testing for 'machine intelligence' before so doing: the Turing Test.

The Turing Test was invented, not surprisingly, by Alan Turing, who was a mathematician greatly involved in computing matters in the mid 1940's. He was very much before his time, and was interested in, amongst other things, the concept of machine intelligence (a biography of Alan Turning has recently been written by Hodges, 1983). The test is as follows (see Fig. 1).

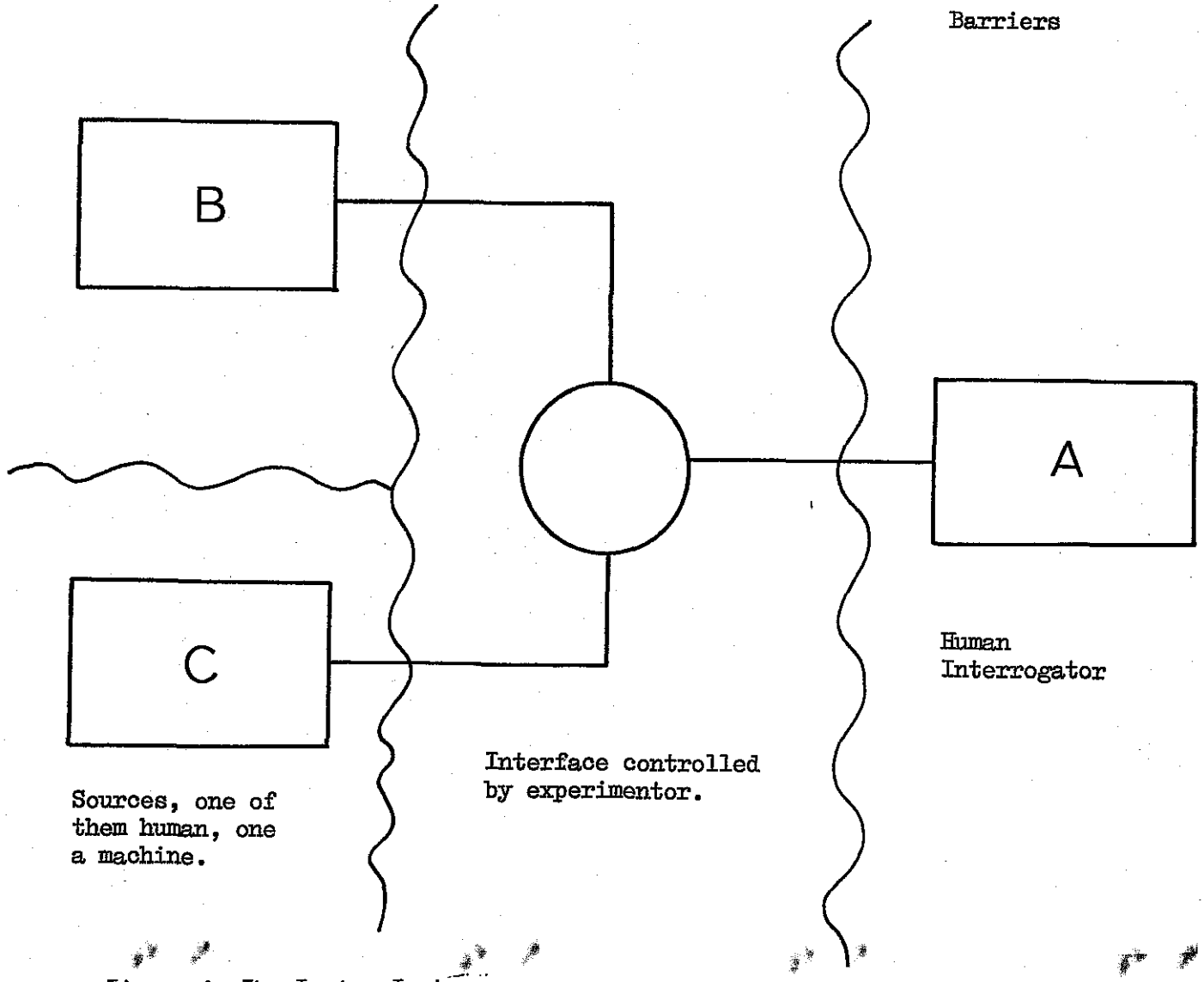


Figure 1. The Turing Test

Human being 'A' is faced with a computer terminal, which he uses to converse with two unknown sources 'B' and 'C'. 'A' is told that one of 'B' and 'C' is controlled by a machine, the other by a human being whom 'A' has never met. An interface is controlled by the experimenter, and is switched between 'B' and 'C' unbeknown to 'A'. If 'A' cannot distinguish between 'B' and 'C' in the course of dialogues with signifiantly better than 50% accuracy, and if this result continues to hold no matter what people are involved in the experiment, then the machine is said to simulate human intelligence.

2.3.1 AI Systems. The following is a list of perhaps the more surprising applications computers have been given by AI researchers. It demontrates very well the type of task involved, and the essentially human qualities that AI systems have (they are in no particular order):

Question Answering in a Story Understanding System (Lehnert, 1977)
Computer-based Medical Consultations (Shortliffe, 1976)
Understanding Natural Language (Winaggrad, 1972)
Using Common Sense (Reiger, 1975)
Solving Calculus Word Problems (Charniak, 1968)
Understanding Children's Stories (Charniak, 1972)
Understanding Physics Problems (Novak, 1976)
Analysing Mathematical Proofs (Bundy, 1975)
Understanding Newspaper Stories (DeJong, 1977)
Writing Stories (Meehan, 1977)
Interpreting and Understanding Cartoons (Adler, 1977)
Analysing and Synthesising Jazz (Ulrich, 1977)
Medieval History (King et al, 1977)
Managing Criminal Court Cases (Buchanan and Fennell, 1977)

2.4 Expert Systems

A branch of AI is the area known as expert systems. It resulted from a feeling by some researchers that the AI aims, as expressed above, were too grandiose and unreachable in the short term. As a result, they started implementing systems which embodied tried and tested AI programming techniques. These systems could display intelligence under a certain set of circumstances, i.e. given enough facts and rules about a specific subject area, and could be used in an advisory or consultative capacity by human beings. These systems were used to try to perform the same tasks as a human expert in a particular subject area, by encoding the expert's way of solving a problem into a set of rules and facts which were 'programmed' into the computer, and which could then be accessed and/or manipulated by a human user in a 'free form' way. (Those familiar with 'conventional' computer systems will be aware that the responses a user gives a computer tend to have to be of a certain kind; i.e. the computer will only recognise a certain number of responses, and badly designed systems may well 'crash' if given an unexpected response. Expert systems, in general, allow a much more flexible approach to user responses, and endeavour to simulate 'natural' conversation.)

To clarify what expert systems are, the following statements are offered as a summary.

An expert system is a computer program which incorporates information about a particular subject area, and the ways in which that information can be used. The result is that a user can use the system in such a way that it appears to have knowledge of that subject area, and can act as an advisor, or consultant or expert in that subject. The important thing about an expert system, and this is something not so far explicitly stated, is that by exhibiting intelligence, it can appear to be making judgemental or evaluative decisions of a type not traditionally associated with computer applications. There is an entire discipline devoted to obtaining information from human experts and encoding it for incorporation into an expert system (known as 'Knowledge Engineering'), and it is the methods by which this information is held and accessed within the computer, together with the programming approach adopted, that makes expert systems the powerful tools they have become.

A more formal definition of expert systems, approved by the British Computer Society's committee of the specialist group on expert systems, is as follows:

> 'An expert system is regarded as the embodiment within a computer of a knowledge-based component from an expert skill in such a form that the system can offer INTELLIGENT ADVICE or take an INTELLIGENT DECISION about a processing function. A desirable additional characteristic, which many would consider fundamental, is the capability of the system, on demand, to JUSTIFY ITS OWN LINE OF REASONING in a manner directly intelligible to the enquirer. The style adopted to attain these characteristics is RULE-BASED PROGRAMMING.'

Does that make everything crystal clear?!

The next chapter is devoted to an examination of how expert systems work.

3. A CLOSER LOOK AT EXPERT SYSTEMS

3.1 What Doesn't Make an Expert System

In a discussion of expert systems it may seem inappropriate to take a brief look at 'conventional' systems; however, it is important that the distinction is made between expert and 'conventional' systems, particularly when dealing with the design of a new system, which will be the case in ITS.

It should by now be apparent that the philosophy underlying expert systems concentrates upon performance i.e. the system is built to provide easy, flexible and natural access to a body of information, or knowledge, which makes up a fairly small subject area. In order to do this, a style of programming is adopted (so-called 'Rule-based'), which is different not only in style and construction, but also in effect.

A small digression is appropriate here. I do not intend to review the whole topic of conventional programming techniques in this report. I could not hope to cover enough ground in the time available either to give someone with no prior knowledge a working understanding of programming techniques, or to give someone with programming experience, but no formal technique training, a similar working understanding. There are numerous texts available on structured programming (which is, in effect, good conventional programming), but the best are perhaps Welsh and Elder, 1982, Page and Wilson, 1982, Knuth, 1973 and Dahl et al 1972. However, it is important to understand the main differences in overall structure between expert and conventional systems, and a small discussion of conventional programming techniques is presented as Appendix A.

The main conceptual difference tends to be one of control. In a conventional system, the user is very much under the control of the program, i.e. it prompts him for more information at certain times, tells him things at certain times, and, generally speaking, will not allow him to re-trace his steps through the program asking for further information. An expert system, again generally speaking, is, in one sense, under more user control, as the user can request information from the system at any time, and does not have to wait to be prompted; in another sense, however, the system is very much in control in terms of 'knowing' which course to pursue. Unlike a conventional system, which tends to

be a sequential branching program of the form shown in Figure 2, an expert system tends to be of the form shown in Figure 3 - not nearly so deterministic.

The structure of expert systems is discussed in the remainder of this chapter, but it can be seen from comparing Figures 2 and 3 that there is a great difference structurally between the two types of system.

The main point to make here is that an expert system is not just

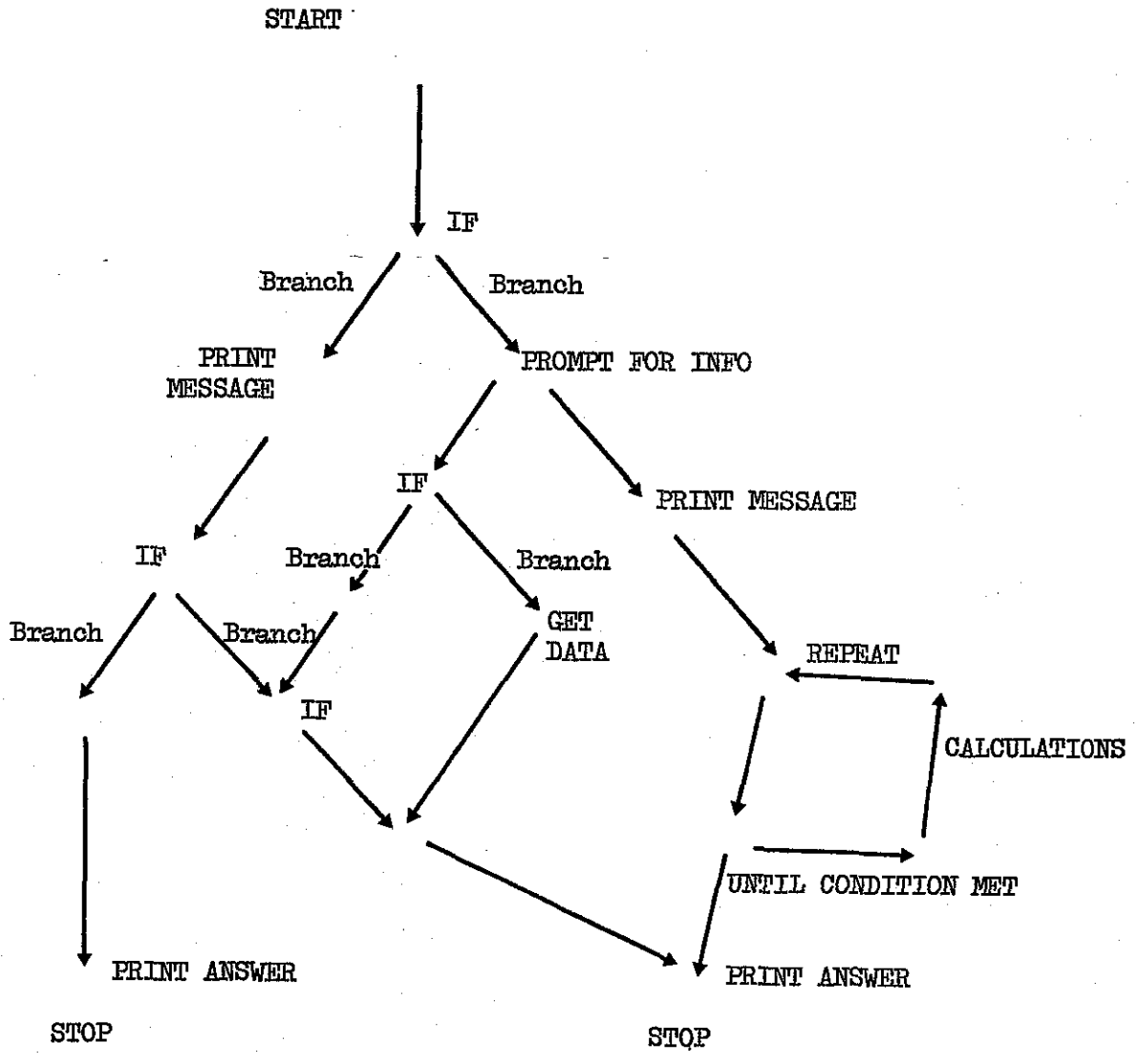


Figure 2 'Conventional' program structure Example

Explanation. This is a diagrammatic representation of a simple, sequential branching program of the type familiar to anyone who has had dealings with computer programming. The main point is that the flow of control is progressively 'down' through the structure, with user messages and input being set at particular points in the program. The whole program is strictly deterministic and, if well programmed, easy to follow.

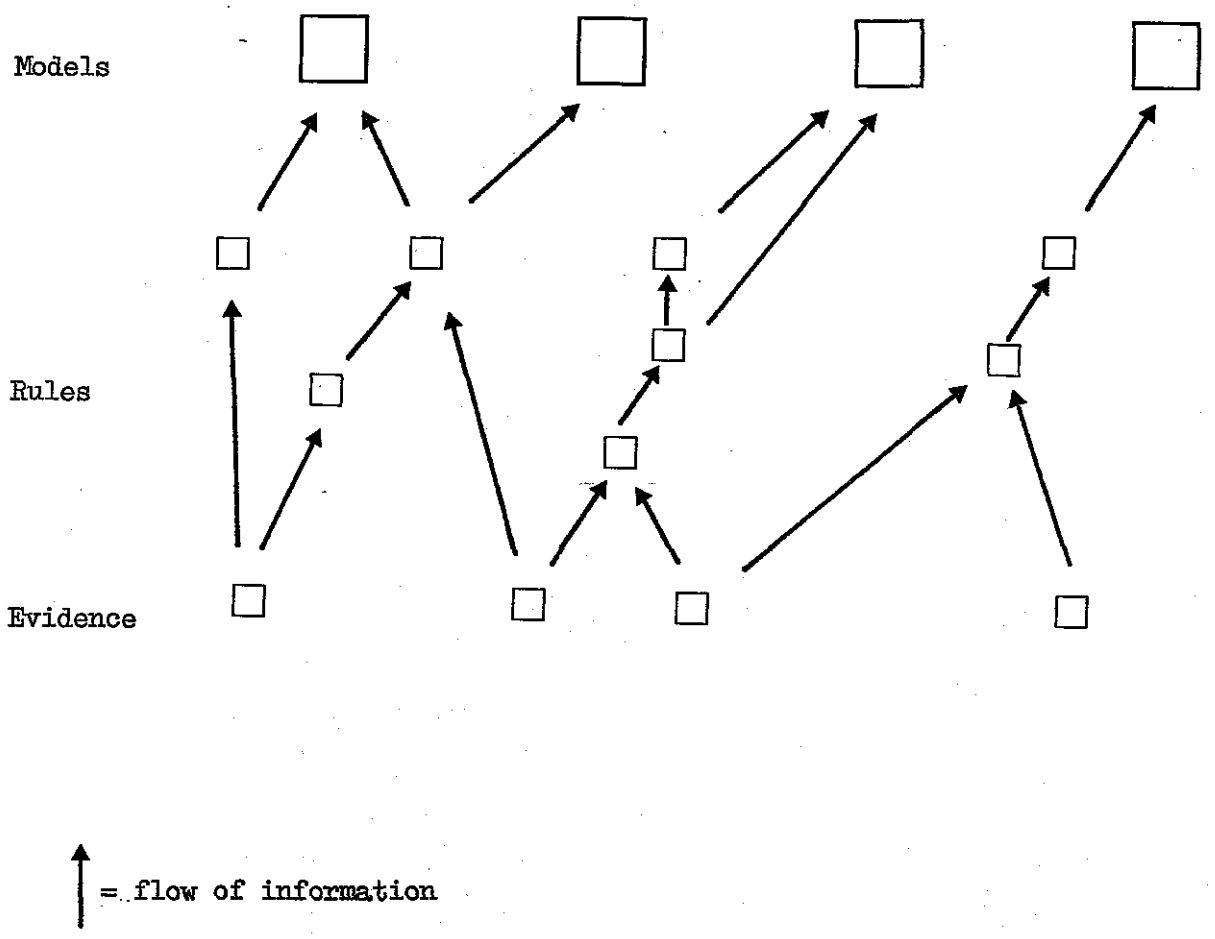


Figure 3 Expert System Structure Example ('PROSPECTOR' style)

Explanation. This illustrates a 'PROSPECTOR'-type expert system structure (see Ch. 4 below for a discussion of 'PROSPECTOR'). It is a network which relates a set of possible evidence to a set of possible models ('models' in this case can be thought of as prototypical situations). Each box represents an hypothesis and the links indicate where the conclusion of one hypothesis forms the input to another, higher, hypothesis. The flow of information is from evidence to model hypotheses. The set of model hypotheses represent the set of 'goals' that the system is attempting to prove.

The evidence hypotheses represent input information, which can be subjective user judgements or sensor measurements or whatever. The network of links between hypotheses represents the rules which, in turn, represent intermediate stages of reasoning.

The system works in the following way: a model is first chosen, either by the user or automatically by the system, and evidence is evaluated by backtracking from the model hypothesis to the evidence. Whenever an evidence hypothesis is encountered, input is requested from the user, or sensors, or whatever. Further models may be evaluated after the first, and, ideally, one model should emerge as the solution to the problem.

a well-designed interactive program; there is a lot more to it than that. A fatal mistake, which people trying to join the expert systems bandwagon tend to make, is to design a user-interface for a previously tried and tested non-interactive program, and call it an expert system! A second mistake, which tends to come in the design stage, is to find an application which requires a computer program, but which is actually a numerical- or algorithm-solvable problem not requiring the 'intelligent knowledge' of an expert system. Kowalski's statement quoted in the proceeding section highlights these problems of terminology, and the deeper one goes into the programming and internal structure of expert systems, the more difficult distinctions are to make.

We shall remain, mostly, at the higher, conceptual, level of the arguments.

3.2 What makes an Expert System

It can well be argued that an expert system is really only a computer program that, when run, makes people say 'I didn't know a computer could to things like that!'. Or, to put it another way, a computer program which does something that you'd have thought would have needed a human expert to do. If this can be borne in mind, then we shall make headway.

So, if we are looking at a computer program that does what a human expert does, then what attributes does it need?

Well, firstly it obviously needs to know something about a subject; we know that that can be done easily enough - a few facts in a database, or perhaps in the form of PROLOG statements (PROLOG is a programming language we'll come to later - if it means nothing to you, don't worry, but go and read Clocksin and Mellish, 1981). So that is alright.

Next, it needs to be able to use the facts it has available to it, it needs to be able to deduce, or infer, further facts from those it has available. We know that this can be done - a set of rules about what action to take if a certain circumstance, or fact, is found to be true is all that is needed. Again, this could be in the form of PROLOG statements, or perhaps just conventional IF statements of the kind:

IF condition THEN action.

So that is alright.

Next, it needs to be able to apply the rules to the facts. How does it know when to apply rules? Well, a human expert works in terms of answering questions, or interpreting information; in other words, he responds to a stimulus of some kind. So, our sytem has to have a way of recognising the appropriate stimulii and taking the appropriate action. This is more tricky, as it has to have an 'interface' designed which will accept stimuli in

the form of questions, or even further information, from a human user (or possibly from a data source such as a traffic counter). The 'user-interface', however, is a common aspect to conventional systems, so we know that that can be done, so that is alright.

A human expert can explain how he reached a decision, and should be able to do so in a way easily understood by the enquirer (although this latter rule is often broken!). Generating explanations is difficult. It has been done in America, but it is recognised as one of the areas in which further work is needed (Kidd and Cooper, 1983). So that is sort of alright - it can be done, but it needs more work.

The other main thing a human expert can do is to weigh up available evidence, and come to a conclusion even if some evidence is missing. This is difficult to put into our system as no-one knows how human beings cope with uncertainty, and how they evaluate partial evidence. Various methods of representing an unknown quantity (both in terms of human processes and evidence!) have been put forward - notably Bayesian Propagation Formalism (Duda et al, 1976); fuzzy logic (Baldwin, 1981); and fuzzy set theory (Zadeh, 1965). All have been incorporated either singly or in concert in expert system designs, so we know that we can at least represent uncertainty, but we are not sure how good that representation is. Evaluations of systems using these uncertainty methods suggest that they can be used satisfactorily (Adams, 1976), but we are still awaiting the definitive method of representing uncertainty.

It should by now be becoming apparent that we are trying to give the computer as many of the facets of a human expert as we can. In order to do this, we need to know how a human expert interacts with both the enquirer and the information he is working on. This is an extremely difficult task, as we would have to have a complete understanding of thought processes in order to endow the computer with the ability to perform exactly as the expert. As stated above, AI is to some extent concerned with building models of thought processes, but the processes themselves are not as yet understood. Expert systems, therefore, tend to try to imitate rather than replicate these processes.

To summarise the requirements for an expert system:

- facts, from which the system can work to infer some higher level information (the database);
- rules, which the system uses to work on the facts. The rules are, in effect, the heuristics the human expert uses when solving a problem (the knowledge source);
- (the database and knowledge source together make the 'knowledge base');
- a method for applying rules to facts, i.e. for selecting a rule under a certain set of circumstances (the inference engine);
- a method of communicating with potential enquirers in as natural a way as possible (the user interface);

- a means of explaining how a particular conclusion has been reached, or why a particular line of inquiry was pursued (the explanation facility);
- a means of representing human uncertainty or incomplete evidence (probability and possibility).

The above requirements are for a 'typical' expert system. Many systems are termed expert by their designers without meeting all, or even any, of these requirements. In fact, there is some argument over just when an expert system is an expert system, particularly when a programming language such as PROLOG is used to build the system. As an example of this confusion, and as an introduction to the next section, let us look at a statement made in the 1983 Expert Systems conference in Cambridge ...

> '... there is a close association between expert systems and rule-based systems. Practically every expert system of significant complexity has been implemented as a rule-based system; and conversely, in North America until recently, expert systems have been virtually the sole application of rule-based languages. In Europe, however, PROLOG has long been used as a rule-based language for a great variety of applications, *such as databases, natural language* processing, computer aided design, compiler writing and rapid prototyping. Many of these applications (rightly or wrongly) would be classified as expert systems had they been implemented in North America'. (Kowalski, 1983)

Just when you thought it was becoming slightly clearer as well!

3.3 Rules and How They Work

For the moment, let us concentrate on rules, what they are and how they work. Much has been made of rule-based programming and rule-based systems, but we haven't, as yet, looked at what they do.

As stated above, an expert system has to be able to infer some information from a set of facts in a database. In order to do this, it has to have some 'procedural knowledge', i.e. instructions to tell it what to do in a certain set of circumstances - these are rules (also termed 'production rules', (Davis and King, 1976)).

The rules represent the knowledge of the human expert, which has been found to be a collection of heuristics rather than an algorithm; i.e. the expert's knowledge tends to consist of 'rules of thumb' developed by experience, and, as such, is difficult, if not impossible, to represent by a sequential algorithm. (See Sloman, 1979 for a discussion of the nature of knowledge.)

So, to describe a subject area, we would have a collection of facts ('*declarative knowledge*' - *describing things*), *together* with a collection of rules ('procedural knowledge' - describing

actions). The interactions of facts and rules (as performed by the 'inference engine') in effect gives us our expert system basis.

The relationships between the inference engine, the database (i.e. the set of facts) and the knowledge source (the set of rules) is as shown in Figure 4 for a 'typical' expert system.

The way in which the system works is, generally, as follows:

The inference engine (i.e. the mechanism for applying rules to facts) has to select an applicable rule from the knowledge source (i.e. the set of rules), and apply it, subsequently updating the database (i.e. the set of facts). This repeats until the solution is reached. It is, in essence, a very simple process - pick a rule according to some criterion, apply it, changing one or more of the facts if necessary, pick the next rule, etc. etc. until the answer, or rather to use the jargon, an assessment, appears.

However, the system has to know how to select a rule, and, which particular rule to select. There are two main methods of rule selection: Data-driven and demand-(goal) driven; they are often used together in expert system implementations.

Rules tend to be of the form:

IF condition THEN action

although implementations of rules in different programming languages can make them virtually unrecognisable. However, think of rules as having the above form.

3.3.1 Data-Driven Rule Selection

Data-driven rule selection methods work as follows:

- the condition part of each rule is hecked against the facts in the database to see if the condition is met.
- a rule is selected whose condition is satisfied.
- the action part of the rule is carried out.
- the facts in the database are updated by the action part of the rule (if appropriate).
- repeat.

If more than one rule's conditions are met in the database, then a rule will be selected either by physical position in the rule set (i.e. first encountered, first used); or by a series of weightings of importance as imposed by the human expert when divulging his knowledge, and encoded as part of the rule structure.

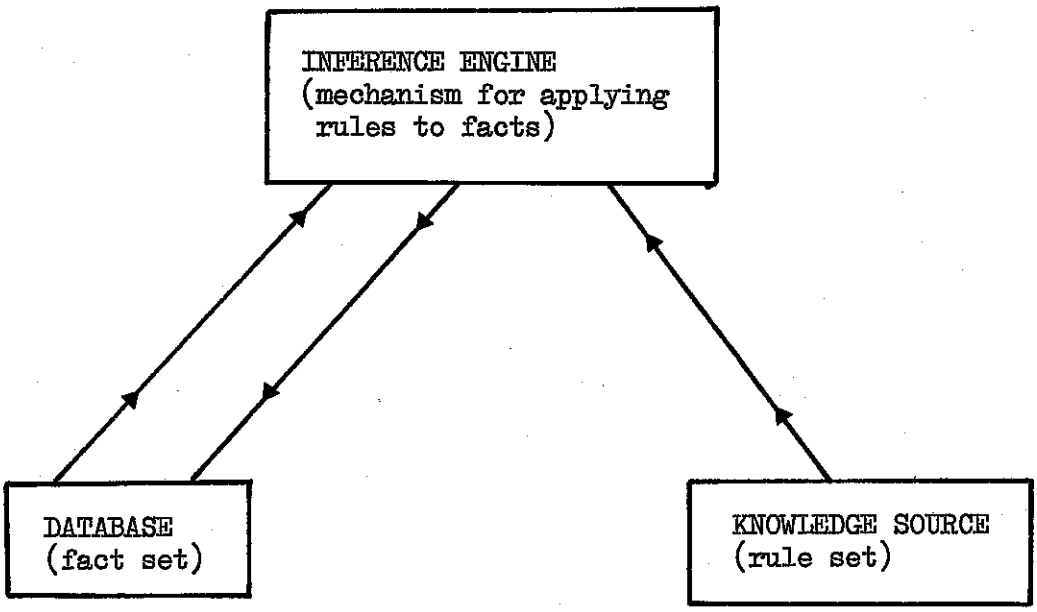


Figure 4 'Typical' Expert System (after Cox, 1984)

The data-driven method is, therefore:

```
Put initial facts into database
Repeat
     Test condition of each rule against database
     Select rule 'R' from the set of possible rules
     Apply rule 'R', updating database
Until database satisfies goal condition
```

An example of an application requiring a data-driven rule selection method might be a real time monitoring and control system for urban traffic. For example, traffic sensors would send information into the database, which would be continually monitored by the application of rules, representing the processes a human system manager would undergo in examining the system state. A 'goal state' for the system would exist, representing optimal system use perhaps, and rules would be applied as a result of data coming into the database and taking the system away from its goal state. Rule actions might well be controlling traffic light times etc. Thus we have a system which operates as a result of the data available to it.

3.3.2 Demand-(Goal) Driven Rule Selection

This approach means that rules are 'chained' together, the action parts of subsequent rules providing information about the condition part of preceeding rules. In our urban traffic network management system example, this approach would be adopted if the expert system had knowledge of, say, bus schedules available to it: suppose that a no. 26 bus were due at a certain point in the network at a certain time, according to the schedules; and that the data being passed to the system by the traffic sensors included a 'bus identifier' of some kind. A demand- or goal-driven rule selection method would examine the incoming data for evidence of the expected bus. System performance could, perhaps, then be assessed on the earliness or lateness of the bus, and appropriate actions taken. The system is taking action to achieve a certain goal.

3.4 How a system knows its subject

The methods of selecting rules (or 'firing' rules to use the jargon) do not, obviously, account for intelligent behaviour, they are purely methods of applying expert knowledge to a particular problem. In fact, expert systems depend entirely upon the knowledge they possess for their intelligent performance. Their power comes from the knowledge base (i.e. the combination of facts and rules), not from the inference engine (i.e. the way rules are applied to facts). Feigenbaum (1979) discusses this in some depth. The dependence upon knowledge gives rise to expert systems' alternative name - Intelligent Knowledge-Based Systems (IKBS). These names seem to be used completely inter-changeably, so we will note the alternative and pass on. (It is, however, worth noting that in some circles at least, an IKBS is only an expert system if it can fulfil a human expert's role; whilst in

others the terms are completely inter-changeable).

How does an expert system get its knowledge? and how is that knowledge represented and used? Well, we have already seen that rules are extremely important, as are facts. Facts will be looked at later, as there are many ways of representing them; let us look first at how the knowledge is initially obtained from the human expert.

3.4.1 What is Knowledge Engineering?

Given that the power of an expert system depends upon the quality of the knowledge it possesses, then it is obviously very important that the knowledge to be used in establishing an expert system is as accurate and well defined as possible. This task of obtaining knowledge from human experts falls to the 'knowledge engineer'.

Knowledge engineering is the term applied to the process of turning the essentially heuristic human expert's knowledge into a set of rules and facts for inclusion in the expert system. There are a number of techniques for eliciting human experts' knowledge, all of them seemingly under continual refinement. They range from the simple, manual method of sitting down and talking with the expert, discussing case studies and working through examples, through to an expert system which infers rules from examples input to it. The former approach is 'explicit' knowledge acquisition, i.e. the expert tries to explain his thought processes, whilst the latter approach is 'implicit' knowledge acquisition, i.e. a computer program extracts the salient features from a series of examples.

Knowledge engineering is a subject obviously closely connected with expert systems, but it is also a large subject on its own account. It has been used to describe the process of building expert systems (Feigenbaum, 1979), i.e. not only eliciting the knowledge, but also designing the data structures and programming requirements for representing the knowledge in the expert system. It has also been used to describe the process of eliciting expert knowledge only (Cox, 1984). In a sense, this conflict of terminology is irrelevant, the essential element is that of acquiring the experts' knowledge in a form that can then be input to the expert system mechanisms.

There are a number of articles relevant to knowledge engineering in both its senses, and it is recommended that any serious potential expert system designer at least read Waterman, 1979; Quinlan, 1979; and the section on Knowledge Acquisition in the Proceedings of the Expert Systems Conference, Cambridge University, 1983.

3.4.2 What is Knowledge Representation?

The method of representing knowledge within an expert system is probably the most argued area in expert systems. We have already

looked at rules and how they work, and systems which rely entirely upon rules, or at least which have a major part of their knowledge represented as rules, are known as 'Rule-based Systems'. Rules are adequate for expert systems which are highly specific and apply to limited subject areas (Cox, 1984); but, when the area is broader, then rule-based representation tends to become unsatisfactory, largely because a lack of 'common sense' tends to become apparant. This is because the set of rules becomes so large that it looses its coherence, and side-effects can emerge that were not anticipated. This problem can to some extent be overcome by adding what is termed 'meta-knowledge' to the expert system (see 3.4.3 below), but it is often more appropriate to adopt a different form of knowledge representation.

It is at this stage that things become a little complicated! There are a number of alternative approaches to representing knowledge, but they seem to resolve themselves into one of two distinct camps. Logic, and its manipulation, termed 'first order predicate calculus', is one (as advocated by McCarthy and followers, see McCarthy, 1979); whilst frame-based representation is the other (as advocated by Minsky and followers, see Minsky, 1975). If you want to read about some of the ways proposed for representing knowledge, then look through the Knowledge Representation sections of the proceedings of International Conferences on Artificial Intelligence. A good general state-of-the-art discussion is available in IEEE Computer, 1983.

Let us look briefly at the two main protagonists.

3.4.2.1 Logic and PROLOG

Logic and first-order predicate calculus are somewhat difficult concepts to come to terms with, and there is little indication in the expert systems literature of any easy introductory text, Crossley et al, 1972 is a good mathematical text for those who are interested. However, the definitive PROLOG text (Clocksin and Mellish, 1981) contains much that is relevant to knowledge representation in logic, and the following discussion relies heavily upon that text. Examples are given in PROLOG form. PROLOG is a computer programming language that is used for solving problems involving objects, and the relationships between objects; it is, in effect, a step towards the goal of programming in pure logic.

A simple example of a fact expressed in logic would be: (in PROLOG form)

```
run-on (trains, rails)                    (1)
```

i.e. 'trains run on rails'

this example shows two important things: firstly, we have defined a relationship 'run-on', and secondly we have defined two objects 'trains' and 'rails'. These are referred to respectively

as a predicate and arguments. Thus we have a predicate 'run-on' with two arguments 'trains' and 'rails'. The selection of names for both the predicate and its arguments is totally arbitrary - we might just as well have said:

a (b, c) (2)

which could have meant the same as (1), but we would have to remember what a, b and c respectively stood for. In effect, we have now defined a fact, i.e. that trains run on rails.

The number of arguments a predicate takes does not seem to be limited, although any given PROLOG implementation will obviously have a maximum number. We could, thus, define further facts:

drive-on (cars, lorries, buses, roads) (3)

i.e. 'cars and lorries and buses drive on roads'.

operate (PTE, buses) (4)

i.e. 'PTE operates buses'.

operate (BR, trains) (5)

i.e. 'BR operates trains'.

operate (private-people, cars) (6)

i.e. 'private-people operate cars'.

etc. etc.

So, we could now put all these facts into a database, and then ask questions about them. The database would look like this:

```
run-on (trains, rails)
drive-on (cars, lorries, buses, roads)
operate (PTE, buses)
operate (BR, trains)
operate (private-people, cars)
```

We could now query this database. PROLOG accepts questions in the same form as facts, preceeded by two special symbols the question-mark (?) and the hyphen (-). Thus:

? - run-on (cars, rails) (7)

i.e. 'do cars run on rails?'

The system would reply: 'No'

but

? - operate (BR, trains) would produce the answer 'Yes'.

If we asked

```
? - likes (HRK, PROLOG)                    (8)
```

the answer would be 'no', because the system can find no reference in the database to any of 'likes', 'HRK' or 'PROLOG'. So, 'no' in effect means 'not as far as I know'. This obviously has implications for mis-spellings etc.

All this is very well, but slightly limiting. How could we ask the system what, for example, BR operates, or what drive on roads. Well, like most programming languages, we use variables. Thus:

```
? - operate (BR, X)                        (9)
```

will produce the answer

```
X = trains.
```

Similarly the query:

```
? - drive-on (X, roads)                    (10)
```

will produce the answers

```
X = cars
X = lorries
X = buses
```

Logic, and its implementation in PROLOG, allows some very flexible and powerful constructs. We can, for example, ask more complicated questions, such as:

```
? - operate (BR, X), run-on (X, rails)     (11)
```

i.e. 'What does BR operate that runs on rails?'

We can also construct rules (hence 'rule-based' programming language) which, in this case, is a sort of generalisation of a list of facts. For example, we could have a database consisting of facts concerning what BR do; say, operate different classes of locomotives, rolling stock, maintenance teams, management teams, etc. etc. looking something like:

```
run-on (class 37, rails)
run-on (freightliner-wagon, rails)
work-in (management-team, headquarters)
etc. etc.
```

We could define a rule to produce listing of all BR mobile equipment, thus:

```
mobile (BR, X):- run-on (X, rails)
```

(the :- reads as 'if').

Hopefully, this somewhat basic discussion will give an indication of the power of both logic and PROLOG. Interested readers are referred to Clocksin and Mellish, 1981 for a far more detailed examination of PROLOG. The use of logic to represent knowledge boils down to defining objects and the relationships between them, and then manipulating those relationships.

A small justification for treating logic and PROLOG together in such a cavalier fashion is required. ITS, whilst most probably using PROLOG for devising rule-based systems defining logical relationships, is most unlikely to want to build complete expert systems from scratch using pure logic as a knowledge representation medium. Thus, we have tried to kill two birds with one stone - introducing both logic and PROLOG.

3.4.2.2 What is Frame-Based Knowledge?

A frame-based knowledge representation medium can be described as follows:

A class of objects or events or scenes is initially described by a prototype (somewhat similar to the SIMULA class concept for those familiar with such things). The prototype is considered as a stereotype description of members of the class. For example:

Suppose we wish to describe a prototype of the Light Commercial Vehicle as specified in TAM, 1981:

```
        PROTOTYPE  :  Light-Commercial
UNLADEN-WEIGHT     :  ¡3 Ton
   REAR-WHEELS     :  Twin                    (1)
         PLATE     :  None
```

Thus, the prototype has 'slots', which are the characteristics of all members of the prototype class, and which in the above example are 'UNLADEN-WEIGHTS', REAR-WHEELS', and 'PLATE' (i.e. presence of rear reflective plate). The values expressed in the prototype slots can act as default values for specific members of the prototype class. The specific members of the class are known as instances of the class, or 'instantiations'. Thus, to represent the fact that a Ford Transit is a light commercial vehicle (if it has twin rear wheels and is ¡30 cwt!), we would write:

```
FORD TRANSIT
INSTANCE OF  :  Light-Commercial          (2)
```

If the system were then asked about Ford Transits, it would say that Ford Transits had an unladen weight ¡3 tons, had twin rear wheels and no reflective rear plate; i.e. the information would come from the values put in the prototype slots.

An important characteristic of frame-based knowledge representation is that of 'property inheritance'. This is where instances of prototypes possess all the characteristics of the

parent prototype. For example, we could change (1) above to include the fact that light commercial vehicles are instances of road-going vehicles, and therefore possess all the characteristics of road-going vehicles such as, say, steering mechanisms, motive power, fuel storage or whatever. Suppose we had specified a prototype 'ROAD-GOING-VEHICLE' with all the appropriate attributes, and wanted the prototype 'light-commercial' to possess all these attributes as well as the further ones we are going to specify, we would write:

```
        PROTOTYPE  :  Light-Commercial
      INSTANCE OF  :  Road-Going-Vehicle
UNLADEN-WEIGHT     :  ¡3 ton                    (3)
      REAR-WHEELS  :  Twin
            PLATE  :  None
```

The system would now know that light commercial vehicles, as well as having the above explicit chracteristics, would also have all characteristics of road-going vehicles, as specified in the Road-Going-Vehicle prototype.

Suppose we now want to make a deduction from the information the system has. For example, suppose we wanted to represent 'George drove a Transit', and 'Norman drove whatever George drove'. This could be represented as follows:

Declare a prototype

```
PROTOTYPE  :  Driving                          (4)
DRIVER     :
VEHICLE    :
```

(note the attributes of 'Driving' are empty, therefore no default values are available)

and two instances

```
D1
INSTANCE OF  :  Driving                        (5)
DRIVER       :  George
VEHICLE      :  Ford Transit

D2
INSTANCE OF  :  Driving                        (6)
DRIVER       :  Norman
VEHICLE      :  Vehicle (D1)
```

Thus, the system could easily give us the answer to the question

'What vehicle did Norman drive?'

As in logic-based knowledge representation, we can construct rules to act as generalisations or to carry out actions. For example, we could build a prototype rule, thus:

```
PROTOTYPE  :  Rule                                    (7)
      IF   :
    THEN   :
```

and an instance of a rule, say

'If x is a transport consultant in firm y, and z is a founder partner of firm y, then z is the boss of x.'

(We use rules in this representation to structure knowledge that is not easily structured by prototypes and their attributes. Data structuring aims to re-create naturally occurring relationships between data objects. See Tenenbaum and Augenstein, 1981.)

```
R1
INSTANCE OF  :  Rule
        IF   :  (x works in y) AND                    (8)
                (z partner of y)
      THEN   :  z boss of x
```

We can incorporate rules into this frame-based representation in such a way that if a slot value in an instance is unknown, then the default value in the prototype can be a call to a rule to infer the required value. For example:

prototypes:

```
PROTOTYPE  :  Consultant                              (9)
     NAME  :
 WORKS IN  :
     BOSS  :  ¡R1¿

PROTOTYPE  :  Partner                                 (10)
     NAME  :
     FIRM  :
```

and instances:

```
B1
INSTANCE OF  :  Consultant                            (11)
NAME         :  E.J. Thribb
WORKS IN     :  Fagin and Partners
BOSS         :

B2
INSTANCE OF  :  Partner                               (12)
NAME         :  W. Shylock
FIRM         :  Fagin and Partners
```

These declarations would mean that the question 'who is E.J. Thribb's boss?' would produce the following actions:

The 'Boss' slot of (11) is unknown, so the default vaue in (9) is implemented. This is a call to rule R1 (8), which then infers

that E.J. Thribb's boss is, in fact, W. Shylock.

There are more complicated constructs of knowledge using this type of representation, but the basic principles remain much the same. See Minsky, 1975 and Nilsson, 1982 for more detailed discussions.

3.4.3 What is Meta-Knowledge?

Meta-Knowledge is the term applied to the knowledge that a system has of its own knowledge. Or, to put it another way, the rules that the system has for asking questions, or activating other rules, etc. For example, a meta-knowledge rule would be of the form:

```
      IF (no. wheels known) AND
         (no. axles known)  AND
         (unladen weight is required)
    THEN (ask question 'n').
```

Meta-Knowledge is one method of giving large, rule-based systems in particular, some degree of 'common-sense'. Another method is by weighting or 'blocking' particular rules. This is dealt with next, and comes under the concept of 'uncertainty'.

3.5 How does a system cope with incomplete and uncertain data?

An important feature of an expert system is its ability to handle uncertainty. Human experts tend to be able to arrive at conclusions after consideration of incomplete evidence, and may often say things like 'its probably factor x that causes the problem', or more precisely 'there's a 75% chance that factor x causes the problem'. Expert systems do the same. We can design our expert system so that if a piece, or pieces, of evidence are missing, the system will still arrive at a valid conclusion, assigning probability to its conclusion.

As stated briefly above, the way in which human beings handle incomplete evidence and uncertainty is, as yet, not totally understood, and so we cannot build an exact model of the human processes into our expert system. There are, however, several theories which address this problem of uncertainty (termed 'possibility' in the jargon), and these are all summarised by Quinlan, 1983. It is argued that all the current forms of so-called possibility theory do not, in fact, work (Cox, 1984), and that the development of a workable theory is the task of the psychologists rather than the computer scientists. The attitude amongst expert system researchers is one of 'we have something that seems to work alright, but it is by no means perfect. We'll stick with it until something better turns up'. The technique that is currently used by most (but by no means all) expert systems is Bayes Theorem, or some derivative thereof. We shall look in some detail at how this is used.

3.5.1 What is Bayes Theorem?

It has been argued that 'PROSPECTOR' handles probabilities and conclusions in a way that is the best worked out of any expert system (Naylor, 1984). It uses Bayes Theorem to assess the effect of a large amount of cumulative evidence on the validity of an hypothesis, and it works like this:

Bayes' formula is applied to prior probabilities to assess the posterior probabilities of an event occurring. An event might be something like the hypothesis that a particular road junction has reached its capacity and needs to be redesigned. Each hypothesis in the system, there can be quite a few, starts off with a prior probability of being true - P(H). Thus, we might give the above example of junction capacity the prior probability 0.5; i.e.

$$P(H) = 0.5 \qquad (1)$$

Now, supposing that there is a huge tailback of traffic at our junction throughout each peak period, the probability P(H) changes to P(H : E), i.e. the probability of the hypothesis given the new evidence. So, the system assigns the value of P(H : E) to P(H), nd we can now go on to look at the next piece of evidence and input that to the system. The problem is how we calculate P(H : E).

The answer is:

$$P(H : E) = LS.P(H)/(P(\text{not } H) + LS.P(H)) \qquad (2)$$

where

$$LS = P(E : H)/P(E : \text{not } H) \qquad (3)$$

which can be roughly explained as follows:

LS is the ratio of the probability of getting a particular piece of evidence if the hypothesis were true, divided by the probability of getting that same bit of evidence if the hypothesis were not true. (Any statisticians will by now have recognised LS as 'the likelihood ratio').

Using the saturated junction example above, we can now see how evidence, or the lack of it, updates prior probability. P(H) was initialised at 0.5; supposing that long tailbacks have a probability of 0.8 if the hypothesis is true, and 0.1 if it is not true. Thus, LS now takes the value 8, because (3) says

$$LS = P(E : H)/P(E : \text{not } H)$$

$$\text{i.e. } LS = 0.8/0.1 = 8 \qquad (4)$$

so, substituting into (2), we get

$$P(H : E) = 8\,P(H)/((1 - P(H)) + 8\,P(H)) \quad (5)$$
$$= 8\,P(H)/(1 + 7\,P(H))$$

Then, substituting P(H) = 0.5 into (5), we get

$$P(H : E) = 8\,(0.5)/(1 + 7\,(0.5)) \quad (6)$$
$$= 4/4.5$$
$$= 0.8$$

so, P(H) is given the value P(H : E), and it is beginning to look as if we need a new junction! Further evidence would then be treated in the same way, updating P(H). However, we have not considered the fact that the evidence we obtain suggests that the hypothesis is not true - so far we have only looked at the 'positive' effects of evidence. We might, for example, find that there are never any queues longer than two vehicles at our junction, suggesting that far from redesigning it, we should perhaps look at adopting that design for all future junctions! How do we put this into our expert system? Well, the calculations are the same, but we are looking at not-E, rather than E, to show that the evidence is lacking. We then calculate the likelihood ratio and update P(H) as before. However, we do need to have a different set of probabilities, as the previous set was based on the presence of evidence, whereas this is now concerned with the effect of a lack of evidence. Thus, we take:

$$LN = P(\text{not } E : H)/P(\text{not } E : \text{not } H) \quad (7)$$

which is the likelihood ratio associated with the lack of a certain piece of evidence. We now calculate P(H) = P(H : not E) as above, but use LN, not LS.

To revert to our junction example, suppose that we have found no evidence of peak period traffic trailbacks, then we might assign the probability that no traffic jams occur given that the junction has reached its capacity as, say, 0.2, and the probability that no traffic jams occur given that the junction has not reached capacity as 0.9. This gives:

$$LN = 0.2/0.9$$
$$= 0.2$$

So, substituting LN for LS, and still assuming P(H) initially as 0.5 we would get

$$P(H : \text{not } E) = 0.2\,P(H)/(1 - P(H) + 0.2\,P(H))$$
$$= 0.1/(0.5 + 0.1)$$
$$= 0.1/0.6$$
$$= 0.1666$$

which indicates that the lack of traffic jams at our junction suggests that we do not need to redesign it.

The above example shows how the presence or lack of evidence can be accounted for in an expert system, and the use of the LS and LN factors prevent the system from asking irrelevant questions; i.e. if you were asked by the system about the presence of traffic jams, and you answered that there was such evidence, the system would then not be so stupid as to ask you if it were true that there were no traffic jams. The LS and LN values (sometimes helpfully described as Logical Sufficiency and Logical Necessiiy) effectively put together the results of the two questions, without actually asking both.

This is still not quite the end of the story, however, because we need to be able to input our evidence in a way that reflects our certainty about its validity. For example, we would recognise no traffic jam, just as we would recognise a 3 mile tailback, but what about a 100 metre queue? or a 4 vehicle queue? (For the sake of example let us pretend we know nothing about transport, and are novice surveyors collecting evidence in a survey.) How do we input a "well, there was a bit of a queue, but not really a traffic jam" statement? For this, indicators of certainty can be used, as described in the next section.

3.5.2 How can uncertainty be represented?

Many expert systems query the user for evidence in the form "How certain are you that ...", and allow answers on a scale -5 to +5, representing complete uncertainty and complete certainty respectively. 0 represents 'don't know'. Some systems treat 0 as a special case representing ambivalence rather than 'don't know, and have a distinct 'don't know' symbol; they also allow 'Yes' and 'No' as completely certain and completely uncertain respectively.

A point to mention here is that experience within ITS has shown that the "how certain are you that ..." phraseology can be misleading. For example, "how certain are you that there is a traffic jam" could be interpreted as meaning either "is there a traffic jam, and if so how big or small on a scale -5 to 5", or "do you think that what you saw constitutes a traffic jam, on a scale -5 to 5". The latter interpretation tends to be the one meant by the system.

We have seen in the previous section how LS and LN are used to weight evidence; we now need to see how our so-called input certainty values are applied to these factors. The relationship between LS, LN and the input certainty is shown in Figure 5 and works as follows:

The 'odds' form of Bayes Theorem is used, whereby

$$\text{odds} = \text{probability}/(1 - \text{probability}) \quad (1)$$
$$\text{probability} = \text{odds}/(1 + \text{odds})$$

Thus, instead of probabilities as in the above section, odds are used, as derived from (1). The principle is the same but odds

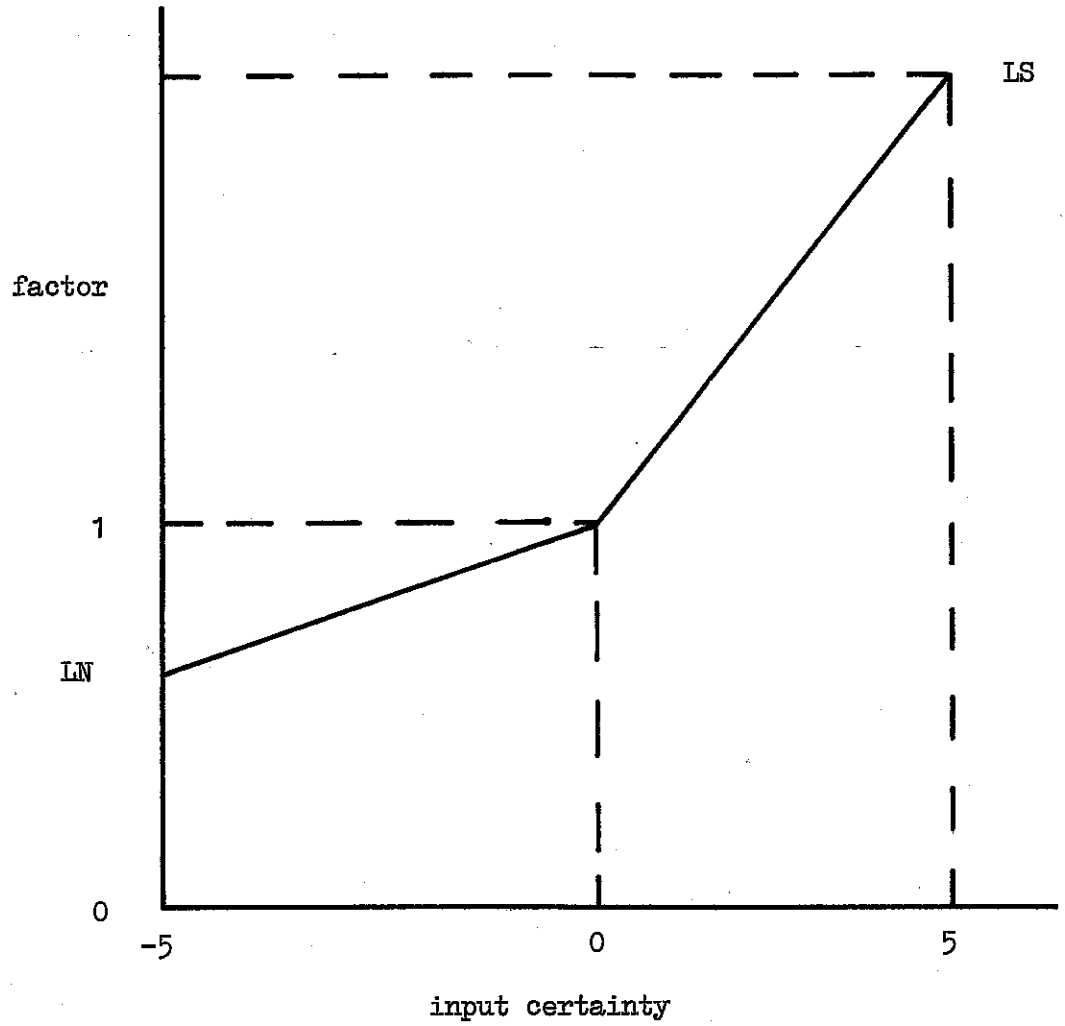


Figure 5 Multiplication Factors for Bayes Rule

seem to be easier to work with. If the input certainty is 0, then the multiplying factor is 1 and the hypothesis prior odds do not change. If the input certainty is less than 0, then the multiplying factor is less than 1, and the hypothesis odds are decreased; and if the input certainty is greater than 0 then the hypothesis odds are increased. The setting of values for LS and LN gives great flexibility in calibrating systems, and allows the expert to put weights on the importance of various bits of evidence.

One other way in which uncertainty in the value of an attribute may be represented is through the use of fuzzy sets - a concept due to Zadeh (1965). This is not explored further here however.

3.6 How a system explains what it does.

In order to communicate solutions to problems effectively, an expert system must be able to offer explanations of what it is doing. Explanations are necessary not only when a solution has been reached, i.e. how that solution was reached; but also when the system requests a piece of evidence from the user, i.e. in understanding why that piece of evidance is needed. The conventional way of including explanatory comments in programs is the so-called 'canned text' method. This is where a comment is printed when the program reaches a certain point in its operation. We should by now, however, be aware that an expert system program does not follow a sequential course of operation, and so 'canned text' would not only be difficult to implement, it would also not give much flexibility.

It is at this stage that we begin to deal with issues not of sole concern to expert systems. The whole area of effective and meaningful dialogue between computer systems and their users is the area known as 'Man Machine Interfacing' (MMI). MMI is another of the concerns identified by the Alvey Report as being in need of further work, and a great deal is currently happening in that area. See Kidd and Cooper, 1983, for an interesting discussion of MMI for an expert system.

So, an expert system needs a method of communicating to its users in a way that is familiar to them, and the most common method of so doing at the moment is to use the phraseology of the rules and facts already in the knowledge base. This presupposes that the user is familiar with the subject area of the expert system, and thus is familiar with the technical terms and jargon of the subject. The system is not being particularly clever in regurgitating explanations in this way - it is really only a slight step ahead of 'canned text'. All that happens when a system is asked 'why' it wants further information, or 'how' it reached a particular conclusion is that it retraces its steps through the rules it has implemented so far.

Returning to our example of vehicle classifications specified in TAM, suppose we have a system which is used to identify vehicle types, and it asks us

'How certain are you that the vehicle had twin rear wheels?'

and we answer

'Why?'

the system would reply something like

> 'Twin rear wheels are a feature of light commercial vehicles, and so we need to establish whether the vehicle had twin rear wheels'

which would come from the rule implemented when asking the question.

The explanation facility of expert systems is still in its infancy, and is mentioned here as a desirable feature of expert systems. The whole area of MMI is as large as expert systems, and the interested reader must follow up such texts as Shortliffe, 1976; Swartout, 1983; and Fox et al, 1983 for discussions of explanatory facilities.

4. SOME EXISTING EXPERT SYSTEMS

This chapter lists a number of expert systems that have been developed in various fields, and gives some details of three of the larger and more well-known expert systems that have been developed and implemented. It then reviews some of those systems that have been designed to enable other expert systems to be generated. This is by no means a comprehensive survey, the purpose being to indicate the range of existing expert system applications. For a good acount of contemporary expert system applications, see Bramer, 1981.

The following list of expert systems in various subjects is based on that in Naylor, 1983. Those described in some detail later are asterisked (*).

Medical Diagnosis	MYCIN*
	PUFF
	PIP
	CASNET
	INTERNIST
Engineering Diagnostics	SACON
	PROSPECTOR*
	DENDRAL*
	SECHS
	SYNCHEM
Circuit Analysis	EL
Genetics	MOLGEN
Mechanics	MECHO
Programming	PECOS
Configuring Computers	R1
Machine Acoustics	SU/X
Medical Measurements	VM
Electronics Tuition	SOPHIE
Medical Tuition	GUIDON
Knowledge Acquisition	TEIRESIAS
	EMYCIN*
	EXPERT
	KAS
Building Expert Systems	ROSIE
	AGE
	HEARSAY III
	AL/X*
	SAGE*
	EXPERT-EASE*
	MICRO-EXPERT*

The systems categorised as 'Building Expert Systems' and, to some extent, those also categorised as 'Knowledge Acquisition', are called 'Expert System Shells'. These are reviewed in section 4.4. But we first describe DENDRAL, MYCIN and PROSPECTOR.

4.1 DENDRAL

4.1.1 History. Initiated in 1965 as part of the Stanford Heuristic Programming Project, in conjunction with the Stanford Mass Spectrometry Laboratory.

4.1.2 Purpose. 'To enumerate plausible structures (atom-bond graphs) for organic molecules, given two kinds of information: analytic instrument data from a mass spectrometer and a nuclear magnetic resonance spectrometer; and user-supplied constraints on the answers, derived from any other source of knowledge (instrumental or contextual) available to the user'. (Feigenbaum, 1979)

4.1.3 Knowledge Representation. Chemical structures represented as node-link graphs; theory of mass spectrometry represented by rules.

4.1.4 Method of Operation. A 3-stage heuristic search - 'plan-generate-test'. 'Generate' is a process for generating plausible structures, under the constraints imposed by the 'plan' process or the user. 'Test' refines the plausible set of structures, discarding the less appropriate and ranking the remainder for user examination. 'Plan' produces direct inferences about likely substructures from patterns in the data indicative of the presence of a substructure.

4.1.5 Comments. Arguably the first major expert system development; currently in every day use by Stanford chemists, it is claimed to out-perform humans in pure structure elucidation under constraints, and to match human expert performnces in structure elucidation with instrument data. DENDRAL was an indicator of the shift in AI towards knowledge-based systems, and has shown the importance of rule-based knowledge representation.

4.1.6 Further Details

Feigenbaum, Buchanan and Lederberg, 1971
Feigenbaum, 1979
Buchanan, Duffield and Robertson, 1971.

4.2 MYCIN

4.2.1 History. Originally the PhD thesis of E. Shortliffe (Shortliffe, 1976), MYCIN has generated an enormous amount of associated work. Many cof the problems encountered (and/or overcome) by MYCIN have been examined by other researchers. The project was carried out in collaboration with the Infectious Diseases group at the Stanford Medical School.

4.2.2 Purpose. The diagnosis of blood and meningitis infections, and the recommendation of drug treatment.

4.2.3 Knowledge Representation. Rules of the form:

IF ... THEN ... with certainty P.

4.2.4 Method of Operation. 'Generation-and-test', similar to DENDRAL, but, where DENDRAL uses heuristic search without feedback, MYCIN establishes a line-of-reasoning by backchaining rules. A number of 'somewhat-true' lines-of-reasoning may be established because each rule supplied by an expert has a degree of certainty associated with it (on a scale of 1 to 10), indicating how confident the expert is of the validity of the rule. MYCIN employs an ad-hoc, but surprisingly workable method of cumulating certainty, and thus is able to handle uncertainty and inexact reasoning. An important point about MYCIN is that its user-interface language is a sort of Doctor-ese English, and so the system appears to be very friendly and understanding; its explanations, as well as its queries, use this language.

4.2.5 Comments. MYCIN is a highly regarded system, not least because its design strategy is very simple. It has led to a number of associated or related system developments, not the least of which are TEIRESIAS, which is a knowledge acquisition sytem, and EMYCIN, which is the shell of MYCIN (i.e. all the elements of MYCIN but without the domain specific knowledge; so the user can input his own domain knowledge whilst still being able to use MYCIN's explanatory, inferential and inexact reasoning facilities).

4.2.6 Further Details.

Shortliffe, 1976
Davis, Buchanan and Shortliffe, 1977
Shortliffe and Buchanan, 1975.

4.3 PROSPECTOR

4.3.1 History. Developed at SRI International, California, the final report being published in 1978 (Duda et al, 1978).

4.3.2 Purpose. A consultant system to aid geologists in assessing how favourable an explanation site or region might be for occurrences of ore deposits of various types.

4.3.3 Knowledge Representation. Inference network of relations between field evidence and geological hypotheses (see Figure 3). Three kinds of relations are used:

i) Logical relations, using standard Boolean logic connectives - AND, OR and NOT.

ii) Plausible relations, using methods derived from Bayesian probability (see Section 3. above), of the form:
IF ... THEN (to degree LS, LN) ...

iii) Contextual relations, used to express any preconditions for using assertions in the reasoning process e.g. there is no point, for example, looking for granite associated minerals if no granite is present in the area.

4.3.4 Method of Operation. As explained in Figure 3 earlier, a 'model' is selected either by the user or automatically (a 'model' in this case refers to 'a body of knowledge about a particular domain of expertise that is encoded into the system and on which the system can act' (Duda et al, 1979), so we are talking about information about each class of ore deposit). The system then backtracks from the model to the evidence, asking for user information, or perhaps sensor information, when an evidence hypothesis is met. The flow of information is from evidence to model, and so the model is evaluated as a result of evidence obtained from the user or sensors. Further models may be evaluated after the first, and so the most likely model can be established from the available evidence.

4.3.5 Comments. PROSPECTOR is apparently in commercial use, its consultation rates being something of the order of $10 per session at commercial computer rates (Duda et al, 1979). The use of an inference network for representing knowledge hs been claimed to aid geologists in their thinking; organising and quantifying expertise for input into a PROSPECTOR model sharpens thinking! (Duda et al, 1979). It is further thought that PROSPECTOR has a potential value as an educational tool.

4.3.6 Further Details

Duda et al, 1976
Duda et al, 1978
Duda et al, 1979.

4.4 Expert System Shells

An expert system shell is a system which has all the methods of inferencing, inexact reasoning, explanation, communicating etc. already established, but which has no domain specific knowledge. To some extent it is comparable to a 'package', such as say, SPSS or SAS, which only needs information from the user in order to work, all the calculating methods and statistical tests are already programmed, it just needs something to work on. Expert system shells, similarly, need something to work, on, and, depending upon the particular system, this will most often be domain knowledge in the form of rules.

An expert system shell is undoubtedly going to be the most appropriate means of developing an expert system in ITS, for reasons discussed in Chapter 5 below but it is difficult to review objectively those currently available, or shortly to become available, to ITS. The reason for this is simply a lack of 'hands-on' experience. We have operated 'Micro-Expert' in its demonstrator form, and, although being impressed by its power, were unimpressed by its user-interface (very unfriendly) and its lack of scope for 'serious' applications - not the least of which is its limited memory availability on a 64k Apple II. These two limitations are serious indeed when developing what will hopefully be a 'showpiece' transport expert system. User friendliness is particularly important as we are not only trying

to convert 'stick-in-the-mud' FORTRAN programmers, but also those practitioners with little or no computing knowledge.

'Expert-Ease' supposedly scores on its distinctly friendly user-interface, and emerges well, generally speaking, from reviews (e.g. SOFT magazine, 1984). It uses Quinlan's algorithm for discovering rules by induction (Quinlan, 1979) for obtaining its knowledge. In other words, rules are induced by the system from a number of examples input to it, and these rules can then be used to solve problems in the same subject area. ITS currently has a bid to obtain 'expert-base' for use on a Sirius micro, for the purpose of evaluation for building transport expert systems.

'SAGE' is a much larger scale system than either of the above, and is well recommended by those in the know. A version is currently mounted on Leeds University's SYSTIME computer, and is maintained by the Computer Based Learning Unit at the University. It is rumoured that there may be a micro-version under development.

Expert system shells are becoming increasingly common as media for developing expert systems, and it is worth looking briefly at some details and applications of expert system shells, particularly if bearing possible transport applications in mind.

4.4.1 Micro-Expert

Micro-expert is produced by ISIS Sytems Ltd., 11 Oakdene Road, Redhill, Surrey (0737 71327/8). It runs under the UCSD - p system, and is really too limited for any serious use. However, it is undoubtedly a good way of becoming familiar not only with some of the operating characteristics of expert systems, but also with the terminology and some of the concepts involved. The manual, when used in conjunction with even the demonstrator system makes a reasonable expert system 'primer'. It is rumoured that a new and 'larger' version of Micro-Expert is currently under development. There has been no reference in the literature to any applications using Micro-Expert, but it is most probable that it is used by many people to come to terms with expert systems before they move to develop 'serious' applications on more powerful expert system shells.

4.4.2 Expert

'Expert' was developed by the Admiralty Surface Weapons Establishment (ASWE), part of the Ministry of Defence (MoD). It is strongly based on 'PROSPECTOR' for its construction, and, from comments in the manual, appears to have been developed so that ASWE, and other MoD establishments, could evaluate the worth of expert systems. It is apparently up and running at Leeds University, again under the auspices of the Computer Based Learning Unit, who are evaluating it themselves. It is thought to be a 'reasonable' system, although it is known to have been modified by Ferranti and CBL are awaiting the arrival of the updated system. It is not known whether any serious applications

have been undertaken using this system. MoD tend not to publicise their new systems!

4.4.3 AL/X

AL/X was developed at Edinburgh University, and is marketed by Intelligent Terminals Ltd. It is a shell, again based on 'PROSPECTOR', and copies of the manual (Paterson, 1981) seem to be hard to find! The only application come across was that reported by Kidd and Cooper, 1983, which was primarily concerned with Man-Machine Interface issues arising from the system. It would be worth investigating AL/X further.

4.4.4 Expert-Ease

'Expert-Ease' is marketed by Expert Software International, 4 Canongate Venture, 5 New Street, Royal Mile, Edinburgh (031 556 3266). It differs from the majority of other systems in that, as stated above, it induces rules from examples. In other words, the user 'teaches' the system rather than programs it with pre-defined rules. It is a 'spread-sheet' system, and is very user friendly. It is the user friendliness that makes Expert-Ease score over other systems, even though it appears to be not as powerful or technically advanced as, say, SAGE. Reviews have generally been favourable, although one particular review compared it unfavourably with a BBC Micro BASIC system called 'HULK', which costs approximately 60 times less! (£25 as opposed to c £1500). ITS will soon be able to make its own evaluation, as 'HULK' is already here, but not working, whilst 'Expert-Ease' is hopefully on its way. Applications apparently include classifying lymphatic caucers; identifying chemicals as potential herbicides, and predicting the Dow-Jones index.

4.4.5 SAGE

'SAGE' is produced by SPL International, Cambridge, UK. It is a larger system than any of the above, and it is not reported that there are any 'micro' implementations, with the possible exception of an ICL PERQ implementation which is, as yet, not commercially available. The SAGE user manual (SPL, 1982) is very comprehensive, and deals with both the structure and the uses of SAGE in some detail. Leeds University's CBL Unit are, again, evaluating SAGE, and a copy of their report would be well worth obtaining. ITS has been advised by CBL that SAGE will probably be the most appropriate shell for developing a transport expert system.

A particularly interesting application of SAGE is that of ICL's Knowledge Engineering Group. This application, known as DRAGON, was the building of a computer sizing consultant system. There are obvious parallels with transport - network sizing and evaluation, assigning flows to branches of the network, cost-effectiveness, etc. etc. It is recommended that the report of the DRAGON system be read, and consideration given to establishing a similar transport project. See Keen, 1983 for

further details.

A further interesting point about SAGE, raised by Keen's report, is that 'conventionally' programmed models or calculations or whatever can be 'plugged into' SAGE, the results being used by the Expert System - a SATURN interpreter? (Note for those not in ITS: SATURN stands for a program developed in ITS, and now widely used in practice, for the 'Simulation and Assignment of Traffic to Urban Road Networks'.)

SAGE is definitely worth further investigation.

5. CONCLUDING COMMENTS

The wide variety of applications and implementations described in Section 4 shows how prevalent expert system thinking is becoming. If we were to widen the constraints on definition to include Decision Support Systems, Data-base management systems, Advisory Systems etc. etc. as is done by some people trying to join the 'expert systems bandwagon' then the lists would be much longer. The difficulties in classifying systems as 'expert' only heighten the definition problems, they do not detract from the value of the techniques.

It is recommended that close liaison be maintained with people using, developing and/or evauating expert systems, as this seems to be the most promising means of obtaining information of direct relevance to ITS as it develops into this field.

For suggestions as to possible transport applications to develop, see the companion report, ITS Technical Note 145.

6. READINGS AND REFERENCES

6.1 Easy Expert System Reading

FEIGENBAUM and McCORDUCK (1984)
MICHIE (1979)
NAYLOR (1983)
COX (1984)
ADDIS (1984)
BRAMER (1981)

Full bibliographic references are given in section 6.2.

6.2 Main references

Adams, D. 'The Hitch-Hikers Guide to the Galaxy', Penguin, 1977.

Adams, J.B. 'A probability model of medical reasoning and the MYCIN model'. Mathematical Biosciences, Vol. 32, pp. 177-186, 1976.

Addis, T.R. 'Expert Systems: An Evolution in Information Retrieval', in Information Technology: Research and Development. C.J. van Rijsbergen (ed.) Vol. 1, no. 4, 1984.

Adler, M.R. 'Computer Interpretation of PEANUTS Cartoons'. Proceedings, 5th IJCAI, 1977.

Alvey Committee, 'A Programme for Advanced Information Technology'. Dept. Industry, HMSO, 1982.

Baldwin, J.F. 'A Theory of Fuzzy Logic' in 'Fuzzy Reasoning and its Applications' (Ed. Mandani, E. and Gaines, B.), Academic Press, 1981.

Baldwin, J.F., 'F.R.I.L. - An Inference Language based on Fuzzy Logic.' Proceedings of Expert Systems Conference, Cambridge University, 1983.

Bramer, M.A. 'A Survey and Critical Review of Expert Systems Research' in 'Information Technologoy for the Eighties', Parslow, R.D. (Ed.), Heyden, 1981.

Buchanan, B.G., A.M. Duffield and A.V. Robertson, 'An Application of Artificial Intelligence to the Interpretation of Mass Spectra' in 'Mass Spectrometry Techniques and Applications', Milne, G.W.A. (Ed.), Wiley, 1971.

Buchanan, J.R. and R.D. Fennell, 'An Intelligent Information System for Criminal Case Management in the Federal Courts', Proceedings 5th IJCAI, 1977.

Bundy, A., 'Analysing Mathematical Proofs (or reading between the lines)', Dept. of AI Research Report 2, Edinburgh University, 1975.

Charniak, E., 'CARDS, A Program Which Solves Calculus Word Problems', MIT Report MAC-TR-51, 1968.

Charniak, E., 'Toward a Model of Children's Story Comprehension', MIT AI Lab. Report AI-TR-266, 1972.

Clocksin, W.F. and C.S. Mellish, 'Programming in PROLOG', Springer-Verlag, Berlin, 1981.

Cox, I.J., 'Expert Systems', Electronics and Power, Vol. 30, no. 3, March 1984.

Crossley, J.N., et al, 'What is Mathematical Logic?', Oxford University Press, 1972.

Dahl, O.J., E.W. Dijkstra and C.A.R. Hoare, 'Structural Programming', Automatic Programming Information Centre, Studies in data processing 8.

Davies, R. and King, J., 'An Overview of Production Systems', in Machine Intelligence 8, pp 300-332, 1976.

Davies, R, B.G. Buchanan and E.H. Shortliffe, 'Production Rules as a Representation for a Knowledge-Based Consultation Program', Artificial Intelligence 8, Feb. 1977.

DeJong, G., 'Skimming Newspaper Stories by Computer', Proceedings of 5th IJCAI, 1977.

Duda, R.O., P.E. Hart and N.J. Nilsson, 'Subjective Bayesian Methods for Rule-based Inference Systems', Proc. National Computer Conference, pp. 1075-1082, 1976.

Duda, R.O., et al, 'Development of the Prospector System for Mineral Exploration', SRI International, Menlo Park, California, Oct. 1978.

Duda, R.O., J. Gasching and P. Hart, 'Model Design in the Prospector Consultant System', in 'Expert Systems in the Micro-Electronic Age', Michie, D. (Ed.), Edinburgh University Press, 1979.

Feigenbaum, E.A., B.G. Buchanan and J. Lederberg, 'On Generality and Problem Solving: a Case Study Using the DENDRAL Program', Machine Intelligence 6, Edinburgh University Press, 1971.

Feigenbaum, E.A., 'Artificial Intelligence Research: What is it? What has it achieved? Where is it going?', Symposium on Artificial Intelligence, Canberra, Australia, 1974.

Feigenbaum, E.A., 'Themes and Case Studies of Knowledge Engineering', in 'Expert Systems in the Micro Electronic Age', D. Michie (Ed.), Edinburgh University Press, 1979.

Feigenbaum, E.A. and P. McCorduck, 'The Fifth Generation. Artificial intelligence and Japan's computer challenge to the world'. Pan Books, London, 1984.

Fox, J., et al, 'Decision Technology and Man-Machine Interaction, the PROPS Package', Proceedings of Expert Systems Conference, Cambridge University, 1983.

Hodges, A., 'Turing', 1984.

Keen, M.J.R., 'An Expert System For Computer Performance Prediction', Proceedings of Expert Systems Conference, Cambridge University, 1983.

Kidd, A.L. and M.B. Cooper, 'Man-Machine Interface for an Expert System', in Proceedings of Expert Systems Conference, Cambridge University, 1983.

King, M., et al, 'Ghosts in the Machine: an AI Treatment of Medieval History', Proceedings 5th IJCAI, 1977.

Kirby, H.R., 'Telecommunications information technology and transport'. Technical Note 134, Institute for Transport Studies, University of Leeds, 1984.

Knuth, D.E., 'The Art of Computer Programming', Vol. 1 Fundamental Algorithms, Addison Wesley, 1973.

Kowalski, R., 'Logic for Expert Systems', Invited Talk, Proceedings of Expert Sytems Conference, Cambridge University, 1983.

Lehnert, W., 'Question Answering in a Story Understanding System', Cognitive Science, Vol. 1, No. 1, pp. 47-73, 1977.

McCarthy, J., 'First-order Theories of Individual Concepts and Propositions' in Expert Systems in the Micro-Electronic Age, D. Michie (Ed.), Edinburgh University Press, 1979.

Meehan, J.R., 'TALE-SPIN, An Interactive Program that Writes Stories', Proceedings of 5th IJCAI, 1977.

Mendelson, E., 'Introduction to Mathematical Logic', Van Nostrand Reinhold, 1964.

Michie, D. (Ed.), 'Expert Systems in the Micro-Electronic Age', Edinburgh Univeristy Press, 1979.

Michie, D., 'The Civilised World of Expert Systems Has Got Steam Fever', Datalink 9, 1983.

Minsky, M, A Framework for Representing Knowledge' in 'The Psychology of Computer Vision', Winston, P.H. (Ed.) McGraw-Hill, 1975.

Naylor, C., 'Build Your Own Expert System', Sigma Technical Press, 1983.

Nilsson, N., 'Principles of Artificial Intelligence', Springer-Verlag, 1982.

Novak, G.S., 'Computer Understanding of Physics Problems Stated in Natural Language', Technical Report NL-30, Computer Science Dept., The University of Texas at Austin, 1976.

Page, E.S., and L.B. Wilson, 'Information Representation and Manipulation Using PASCAL', Cambridge University Press, 1982.

Paterson, A., 'AL/X User Manual', Intelligent Terminals Ltd., Oxford, 1981.

Quinlan, J.R., 'Discovering Roles by Induction From Large Collections of Examples', in 'Expert Systems in the Micro-Electronic Age', D. Michie (Ed.), Edinburgh Univeristy Press, 1979.

Quinlan, J.R., 'Inferno: a cautious approach to uncertain inference', Computer Journal, No. 26, 1983.

Rieger, C., 'The Common Sense Algorithm as a Basis for Computer Models of Human Memory, Inference, Belief and Contextual Language Comprehension', Proceedings TINLAP Workshop, MIT, 1975.

Shortliffe, E.H. and B.G. Buchanan, 'A Model of Inexact Reasoning in Medicine', Mathematical Biosciences 23, 1975.

Shortliffe, E.H, 'Computer-based Medical Consultations: MYCIN', American Elsevier, New York, 1976.

Sloman, A., 'Epistemology and Artificial Intelligence' in 'Expert Systems in the Micro-Electronic Age, Michie, D. (Ed.), Edinburgh University Press, 1979.

SOFT magazine, 'Your Specialist Subject?', Feb/March 1984.

Special Issue on Knowledge Representation', IEEE Computer, 1983.

SPL 'SAGE User Manual', Systems Programming Ltd., SAG03, 1982.

Swartout, W.R., 'XPLAIN: A System for Creating and Explaining Expert Consulting Programs', Artificial Intelligence 21(3), 1983.

Tenenubaum and Augenstein, 'Data Structures Using Pascal', Prentice Hall, 1981.

Traffic Appraisal Manual (TAM), London, Dept. Transport, 1981.

Ulrich, J.W., 'The Analysis and Synthesis of JAZZ by Computer', Proceedings 5th IJCAI, 1977.

Welsh, J.R. and J. Elder, 'Introduction to PASCAL', Prentice Hall 1982.

Wigan, M.R., 'Information technology and transport: what research needs to be started now?' Working Paper 172, Inst. Transp. Stud., Univ. Leeds, Leeds, 1983a.

Wigan, M.R. 'Expert systems and Prolog course'. Extracts from a report to the Australian Road Research Board. Technical Note 133, Inst. Transp. Stud., Univ. Leeds, Leeds, 1984.

Winograd, T., 'Understanding Natural Language', Academic Pres, New York, 1972.

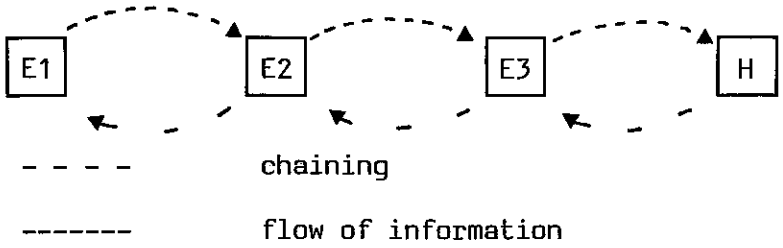
Zadeh, L., 'Fuzzy Sets', Information and Control 8, 1965.

7. GLOSSARY

Artificial Intelligence (AI): the study of the construction of intelligent artifacts, and the development of principles, methods and techniques useful in such construction. Also seen as a means of seeking explicit and valid information processing models of human thought.

Back Tracking: Another term for Backward Chaining.

Backward Chaining: The common term for Back Tracking and the Goal-Driven Rule Selection Strategy. Involves the system first considering hypothesis H, discovering it needs to know evidence E3 to establish H, discovering it needs to know E2 to know E3, and E1 to know E2, so it requests data on E1, after which it proceeds forward to H.



Bayes' Theorem A method of dealing with uncertainty and incomplete evidence.

$$P(H:E) = \frac{P(E:H)\ P(H)}{P(E:H)\ P(H) + P(E:\text{not } H)\ P(\text{not } H)}$$

The probability of the hypothesis being true given the evidence.

Certainty: A means of accepting user input, usually on a scale of -5 to +5, meaning completely uncertain and completely certain respectively. 0 usually means 'do not know', but can be treated differently by some systems. Questions are asked of the user in the form:

"How certain are you that ....."

Certainty Factor: A value associated with a rule, showing how much confidence the expert who supplied the rule has in its validity. Usually on a scale from 1 to 10. Particularly associated with the MYCIN system. Rules with certainty factors are of the form:

IF ... THEN ... with certainty P.

Database: The area of memory in which all the program's variables are contained. Initially, it represents the initial data or facts from which the expert system is to infer some higher-level information.

Data-driven Rule Selection Strategy: Another term for Forward Chaining.

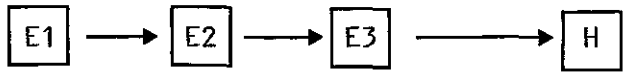
Domain Area: The subject area of an expert system. For example, the domain area of MYCIN is blood and meningitis infections and associated drug treatment. A *limited domain expert system*, therefore, would be only working in a very small subject area.

Expert System: The formal definition from the British Computer Society's committee of the specialist group on expert systems is:

> "...the embodiment within a computer of a knowledge-based component from an expert skill in such a form that the system can offer Intelliegent Advice or take an Intelligent Decision about a processing function. A desirable additional characteristic, which many would consider fundamental, is the capability of the system, on demand, to justify its own line of reasoning in a manner direclty intelligible to the enquirer. The style adopted to attain these characteristics is rule-based programming."

Explanation facility: the means whereby a line-of-reasoning, or a conclusion, is explained to the user. Usually it involves re-tracing the steps the system has taken so far, using the text of the rules fired as explanatory text.

Forward Chaining: The common term for a Data-Driven Rule Selection Strategy. It simply means that the system is given evidence, after which it can deduce the validity of an hypothesis. i.e.



For example, the system is given E1, after which it is given E2, then E3, after which it can deduce H.

Frame-Based Knowledge Representation: A method of representing information and the relationships between the bits of information. A class of objects or events or scenes is initially described by a prototype, which has attributes; an example of a class is an 'instantiation', and can take the values of the prototype's attributes (known as 'Property Inheritance'), as well as having its own attributes. Essentially a data-structuring approach to representing knowledge.

Fuzzy Logic: Another method of dealing with uncertainty and imcoplete evidence; in conjunction with fuzzy set theory it is used to attempt to resolve ill-defined concepts. The example given in a paper on the subject is 'a bush, for example, cannot be precisely defined. Our ability to decide if a given object is

a bush or a tree is not the result of a lack of information but a lack of definition' (Baldwin, 1983). Fuzzy logic and set theory attempt to resolve such problems.

Fuzzy Set Theory: See above.

Goal: A terminating condition for an expert system insofar as it tries to attain goals, which involve proving and/or disproving hypotheses. A goal state is the set of values in the database that the system is trying to obtain.

Goal-Driven Rule Selection Strategy: Another term for Backward Chaining.

Inference Engine: The mechanism by which inferences are made, rules fired and deductions made. The old philosophical term for a computer.

Inference Network: The collection of rules, models and evidence hypotheses that together make up the inter-relationships enabling the system to progress through its stages of reasoning.

Instantiation: An instance of a class in a frame-based knowledge representation; or more generally, the assignment of a particular value to a variable in a program.

Intelligent Knowledge-Based System (IKBS): A system which uses knowledge, either frame-based, logic-based or rule-based, or any combination, to exhibit 'intelligence'. 'Intelligence' has yet to be defined by psychologists, but it is taken here to mean that the system appears to understand the particular domain area. Can be used as an alternative name for an expert system, but does not necessarily exhibit expert performance.

Knowledge Acquisition: The process of eliciting a human expert's knowledge. The term 'knowlege engineering' is sometimes applied to this task; although it most often refers to the whole task of eliciting the information, and then coding it into the appropriate knowledge representation form. Knowledge acquisition can be done by humans in conversation with an expert; interactively by computer, such as the TEIRESIAS program; or automatically by computer using induction from examples.

Knowledge Base: This is the complete set of facts (declarative knowledge) and rules (procedural knowledge) which together constitute the computerised version of the human expert's knowledge. i.e. the knowledge source and the database.

Knowledge Engineering: Used to mean either the knowledge acquisition task, or the complete field of building an expert system.

Knowledge Representation: The methods by which a human expert's knowledge is represented within the computer. There are many different types of knowledge representation, but they can

generally be classified into either logic-based or frame-based.

Knowledge Source: The collective term applied to the part of the system which contains the necessary information to solve a particular problem, i.e the rules for interpreting the facts.

Logic-Based Knowledge Representation: A method of representing objects and their associations. First order logic is used, incorporating set theory, as an alternative to frame-based knowledge representation methods. Predicate Calculus is used to manipulate the objects and their associations.

Man-Machine Interface (MMI): The way in which the user and computer communicate. Much research is currently being carried out into designing MMIs which are as natural as possible.

Meta-Knowledge: The knowledge that the system has of itself, i.e. a set of rules telling the system which evidence to acquire next given the evidence it already has.

Model: A body of knowledge about a particular domain of expertise that is encoded into the system and on which the system can act. Refers to a 'prototypical' situation - a 'best-possible' situation.

Natural Language: Familiar language to humans. MMI work tends to concentrate on designing natural language interfaces which would enable users to communicate with the computer in, say, English or French, instead of 'computer-ese'.

Pattern Matching: Self-explanatory. Used in PROLOG for answering queries. When a question is asked of PROLOG it searches through the database looking for facts that match the situation part of the rule used as the question, or the fact part of the question. For example, suppose a database consists of the facts:

```
drive (cars, people)
walk-on (paths, people)
fly-in (planes, people)
```

we could ask

```
? - fly-in (paths, people)
```

and PROLOG would try to match the pattern of the question against the facts in the database, replying 'no'.

Predicate Calculus: A form of logic used to represent objects and associations, and the means of changing the inter-relationships. Objects are represented by 'terms' which can be either 'constant symbols', 'variable symbols' or 'compound terms' (the latter being a type of structure). See Mendelson, 1964 for a detailed treatment.

Production Rules: Rules of the form:

IF conditions THEN actions

often used to encode human experts' heuristic expertise.

PROLOG: A programming language which is essentially rule-based. Known as a 'very high level language' or a 'functional language'; 'conventional' languages such as PASCAL, ALGOL etc are 'high level languages' or 'imperative languages' (the latter owing to their reliance on statements). See Clocksin and Mellish, 1981 for the definitive text on PROLOG.

Rules: can be thought of as a series of IF... THEN... statements. See Production Rules.

Rule-Based System: A system which relies upon a collection of rules for its operations. Unlike IKBS in that although the rules often represent knowledge, they are the only type of knowledge representation in the system.

Rule-Firing: The term for activating a rule.

APPENDIX A: 'CONVENTIONAL' PROGRAMMING

A1. Programming Techniques

Writing computer programs is an art-form (Knuth 1973) however, thre are a number of techniques that can be applied when writing a program that can assist in making programs correct and comprehensible. An interesting starting point is a set of figures produced by IBM in the late seventies (Source unknown). These figures show the amount of time each stage in the software development cycle takes, and highlights two very important points:- firstly, the steps involved in writing software; and secondly the inordinate amount of time involved in maintaining software.

Software Development Cycle	% Time
What is Required	9%
Specification of System	3%
Design of Program	5%
Write Program Code	7%
Test Program Code	8%
Complete System Testing	7%
Maintain System	67%
	100%

The point that must be taken from these figures is that programs should be easy to read - the easier a program is to read, then the less time is spent by people trying to understand what the program actually does (as opposed to what the documentation says it does), and hence the less time is spent in unnecessary, and costly, repetitive work.

The remainder of this section looks at some techniques for writing 'good' programs, but as a general rule always bear in mind the following good programming practices:

- Use a structured high-level language (e.g PASCAL. FORTRAN is unstructured, outdated and error prone, BASIC is not considered to be a programming language!).
- Use identifier names that mean something (e.g. call the variable for 'the next car in the queue' next-car-in-q, not "nq").
- Use sparing comments (your code in PASCAL for example, should be clear).
- Use small procedures (there is hardly ever any need to have procedures/subroutines larger than c 20 lines).
- Specify your procedures in English (i.e. the main action(s), parameters in and out).

- Use simple algorithms (do not write tortuous algorithms for the sake of it. Your might gain as much as 20 instructions - which on the Amdahl might save 20 miliseconds CPU time - but you may well spend 2 hours trying to understand how the algorithm works when you come back to it in 6 months time).

- Document everything.

- Review your program design when you have finished - you may not be able to change this program, but you will almost certainly have learnt something of use to your next program.

A2. Structured Programming

This is often (incorrectly) called 'GOTO-less programming'; however, there is only one reason for using a GOTO statement, and that is to pick up a catastrophic error which requires a program stop. For example:

```
IF Catastrophic_Error THEN GOTO Emergency_Stop
```

In all other cases where a GOTO would be used, then use the language constructs provided; if, as in the case of FORTRAN or BASIC, there are few constructs provided, then build some out of the language primitives. Always take full advamtage of the facilities a language offers.

There are two main reasons for avoiding using the GOTO: firstly it complicates the compiler; and secondly it tends to complicate programs - the reader has to understand a larger portion of the program than otherwise.

In summary, follow a sequential course through the program, use procedures/subroutines/functions to solve distinct problems, do not 'dot around' the code, use the language constructs provided.

A3. Program Development Design Strategies

Structured programming conjectures that confidence in the correct behaviour of a program is most easily attained by a well-organised program development process, in which each step requires only a relatively simple justification. There are two such design strategies: the top-down approach and the bottom-up approach.

A3.1 Top-Down Development

The top-down approach embodies the following steps:

- Develop a program in a number of small design steps, starting from the program specification and ultimately finishing with an implementation.

- Make each step decide very little about the development of the program.

- Delay every design decision for as long as possible.

- If each step forms a number of sub-problems, then design them top-down as well.

- Use procedures and functions to hide 'problems', i.e. a procedure or a function should solve at most one specific problem.

A3.2 Bottom Up Development

Unless one is working in assembler on a completely 'empty' computer, then a number of routines will exist already. Make use of those to build slightly higher level routines and in turn use these to build even higher level routines. For example, use a language defined function, such as '*' (multiply), to build '**' (to the power of).

As a general rule, use top-down for program design, and bottom-up for program coding.

A4. Testing Software

Testing can only show the presence of bugs, it can never show that no more bugs remain, even after testing bugs may still exist. So, what is a 'correct' program. The following criteria together define a correct program.

1) A program that contains no syntax errors.
2) A program that contains no compilation or execution errors.
3) There exists test data for which the program gives correct answers.
4) For typical sets of test data the program gives correct answers.
5) For difficult sets of test data the program gives correct answers.
6) For all possible sets of data which are valid for the problem specification, the program gives correct answers.
7) For all possible sets of valid test data and all likely conditions of erroneous input the program gives correct answers.
8) For all possible input the program gives correct output.

When choosing test data, try to follow these guidelines:

i) use simple cases
ii) use extreme cases, i.e. test the 'edges' of valid ranges
iii) use special values
iv) make sure all possible branches of flow are followed
v) remember to write down what you expect the results to be.

A5. Data Structuring

Virtually all high-level programming languages provide data structuring capabilities, ranging in complexity from the simple 'n' dimensional array in FORTRAN, through the pointer and record construction of PASCAL, to the abstract data type capabilities of languages such as CLU. If is important when designing a program that organisation of data is taken into account, and that, wherever possible, correct structures are employed. Much storage space can be wasted by, for example, programming stacks and queues as arrays rather than linked lists. In transport, probably the greatest offender is the 0-D matrix, particularly when sparse. A glance through Tenenbaum and Augenstein, 1981 will show how efficiently this can be done! There is not really space here to discuss data structuring in any depth, suffice it to say that this is an essential part of program design, and should always be one of the major decisions when designing a program. If a language is lacking in data structuring capabilities, then consider using a different language.