A Practical Introduction to Rule Based Expert Systems

Sasikumar Mukundan
Centre for Development of Advanced Computing

Srinivasan Ramani
International Institute of Information Technology Bangalore

Raman Chandrasekar

A Practical Introduction to Rule Based Expert Systems

M Sasikumar
S Ramani
S Muthu Raman
KSR Anjaneyulu
R Chandrasekar

Published by:
Narosa Publishing House, New Delhi

Trademarks

Preface

Have you read a book from beginning to end and then realized that none of the ideas in the book can be directly used in the real world? This may be acceptable in some fields, but not in the area of expert systems, which is inherently practical. You cannot really understand what expert systems are unless you have implemented one.

In writing this book we have tried to ensure that readers get *practical* knowledge, which will help them not only to understand expert systems technology, but also to implement expert systems.

We have been teaching courses on expert systems for many years. We have evaluated many books for use with these courses. We found that most books currently available do not provide a satisfactory introduction to expert systems. Books we looked at were either too full of detail, or too abstract. Basic concepts were completely left out in some books.

There are some basic ideas in expert systems which one should understand before implementing expert systems. However, these ideas should only be the stepping stones to the practical aspects of expert systems technology. Keeping this in mind, we have tried to provide a judicious mixture of principles and practical aspects in this book.

The book is structured around a part-time course on expert systems which is conducted at the National Centre for Software Technology (NCST). The emphasis in that course, and in this book, is on rule based expert systems. However, aspects related to other types of expert systems have also been covered.

Chapters 1 to 6 deal with a variety of issues related to the principles of expert systems. Chapter 7 describes some rule based expert systems in use. Chapter 8 provides advice on knowledge engineering, and Chapter 9 on the verification and validation of rule based expert systems. Chapter 10 deals with some other topics of interest in the area of expert systems. Chapters 11 to 13 are case studies which illustrate techniques which can be used in developing rule based expert systems. Complete rule bases for these case studies are listed in the appendices. The expert systems background required for the case studies is covered in the first eight chapters.

Every chapter has a summary, which lists the key ideas discussed in that chapter. There are exercises at the end of most chapters. Answers to selected exercises are given in an appendix.

There are a few sections in the book which may be omitted during the first reading. These sections are marked with an asterisk (∗).

The book does not require any computer science or artificial intelligence background. The book can be used as a textbook for a semester long course on expert systems or as a supplementary book for courses on artificial intelligence. It would also be of use to anyone who is interested in implementing expert systems.

It would be valuable to combine practical work using an expert system shell (such as Vidwan, a shell developed at NCST), with the study of this book. While the case studies and examples used in the book are primarily based on Vidwan, they can easily be converted to run on other rule based expert system shells.

The words *he*, *him* and *his* are used throughout the book for convenience; they do not indicate any gender bias.

We thank the National Centre for Software Technology for providing facilities to create this book. We would also like to thank the Knowledge Based Computer Systems (KBCS) project which funds Artificial Intelligence research work at NCST. The KBCS project is funded by the Government of India with assistance from the United Nations Development Programme.

We thank Ms Madhavi Salvi for helping us with the edits which were done on the chapters. We would also like to thank our colleague Mr TM Vijayaraman who helped us with the use of LaTeX in the preparation of the book.

Versions of this book have been used in courses at NCST during 1991 and 1992. We thank the participants of these courses for their feedback and comments.

M Sasikumar
S Ramani
S Muthu Raman
KSR Anjaneyulu
R Chandrasekar

Contents

1. Introduction

1.1 Artificial Intelligence and its Sub-areas

The creation of computer programs which exhibit some features of human intelligence has been the goal of Artificial Intelligence (AI). Artificial Intelligence has a number of important sub-areas such as natural language understanding, computer vision, theorem proving, understanding spoken utterances, game playing and so on. These areas pose varied challenges to those who create AI systems.

The human brain itself seems to have different ways of dealing with work of these different types. For instance, processing visual information involves the handling of millions of bits of information, usually in a parallel fashion. Millions of nerve fibres carry what the eye sees, to the brain. The brain carries out a lot of parallel processing to recognize lines, corners, etc. It performs specialised primitive operations on pictures represented as two-dimensional arrays of picture elements.

We must compare this type of computation with, for instance, the work involved in solving what are called word problems. One sees this kind of problem in school books, expressed in natural language, but it requires some mathematical skill to solve them. For instance, let us look at the following problem:

> *Vivek is six years older than his sister.*
> *Three years ago, he was double her age.*
> *What is his sister's age?*

The computational tasks involved in this seem to be quite different from the ones involved in computer recognition of visual patterns. Herbert Simon, an AI pioneer, has expressed his belief that problem solving efforts such as the one demanded by word problems may not involve any significant parallel computing.

Creating computer programs to prove mathematical theorems appears to be yet another type of task. A concern with the nature of mathematical and logical reasoning guides work in this field. Partly because most of the researchers involved in this field are often professional mathematicians or logicians, formal methods are more frequently used in this area of work. Many believe that the kind of common sense reasoning necessary for solving word problems would be valuable in theorem proving also.

These tasks illustrate the richness and complexity of the tasks that AI has set for itself.

1.2 Topics in AI

It is worth listing some of the areas of research in AI:

- natural language understanding
- knowledge representation
- common sense reasoning
- logic in AI
- intelligent tutoring systems
- planning and scheduling
- computer vision
- robotics
- speech understanding
- machine translation using AI techniques

This list is not complete in any sense. But it does capture some of the important areas of AI activity. We shall not attempt to discuss all of these topics. But it is worth mentioning why some of the topics are of great interest.

If language is a vehicle of thought, it is natural that language understanding is given considerable importance. The complexity of human language is, one would suspect, related to the complexity of the world that one has to represent in one's mind. So, issues of language understanding and knowledge representation have a common ground.

Work in scheduling is creating a new form of operations research. A high volume of information is used by human schedulers in carrying out their job. For instance, in scheduling aircraft, you might have to take into account the fact that flying crew should not work more than 10 hours at a stretch. You might have to remember that a certain airport is closed for traffic between 11 PM and 6 AM. There may be thousands of items of such knowledge. Hence, knowledge based scheduling, for creating an airline schedule, makes a lot of sense.

We could have said of early Computer Assisted Instruction (CAI) systems:

"Forgive them, for they know not what they are programmed to teach!"

Similarly, we could have said of early machine translation systems:

"Forgive them, for they do not understand what they are expected to translate!"

Researchers are now working on Intelligent Tutoring Systems and on machine translation systems which have capabilities to "understand" the words and phrases they are using. Obviously, a teacher or a translator who understands what he is saying is better than a simple machine which spews out words in an unthinking way. AI based approaches to old problems are, therefore, more promising than any approach tried so far.

1.3 The Turing Test

Alan Turing, the pioneer of computing in many ways, wrote a seminal paper [Turing, 1950] in which he proposed a fair and reasonable test to identify what is an *intelligent* computer system. It is intelligent behaviour that characterises an intelligent system. Whether it is implemented in silicon, or in flesh and blood, should not be the criterion to decide if something is intelligent. Similarly, it does not matter if it was created by biological processes or by computer programming. The common argument that "a man made it, so it cannot do better than its creator" is patently wrong. Man constantly builds machines that outperform him in physical strength, speed, precision, etc. Why should this not be the case with information processing machines also?

Let us, therefore, proceed to visualise a fair test for an intelligent machine. You communicate with it through a computer terminal and forget about its physical nature. Does it behave like an intelligent person? For instance, can it carry out a conversation such as the one in Figure 1.1?

Q: Hello! How are you?

A: Fine, thank you. Are you a visitor here?

Q: Yes. Can you tell me where I can get a newspaper?

A: You have to go out to the main road, and it could take quite a while. Why don't you use the "daysnews" utility and read wire service news on your terminal?

Q: Is that service available on this machine?

A: Yes, it is. In fact, if you are at a graphics workstation, you can even see the news photographs of the day.

Q: Thanks. But, tell me, is there anything interesting in the news today?

A: Well, if you like sports, Andre Agassi has won the Wimbledon Cup.

Figure 1.1: The Turing Test

Can we create a computer program that converses like this? Note that the program would have to have a number of capabilities. It has to understand a human language, in this case English. It has to have some idea of who might be talking to it, and know about the person's possible interests. In the example shown above, it seems to know a bit about the geography of the campus and about what is in the wire service news.

With some effort, you can create an example where such knowledge is less important for keeping up an intelligent conversation. But our purpose was to highlight the relevance of *knowledge* along with intelligence. What is the relationship between knowledge and intelligence? Can the availability of knowledge to a computer program make intelligent behaviour more practical? We shall return to this theme soon.

1.4 AI and General Problem Solving

Early attempts in creating AI systems placed a lot of emphasis on general problem solving [Newell and Simon, 1972]. We can surmise that this was partly in defence against the argument that computer programs of the time could handle only very specialised tasks. Problem solving methods used in tasks such as chess playing did not resemble the more general methods used by human beings. We seem to use the same principles in finding our way in a new city, and in tracing the wiring connections on a printed circuit board. But a computer program can do far better than us in optimising the layout of conducting patterns on a printed circuit board, but not be able to do anything about a city map.

AI pioneers, therefore, focussed attention on general principles on which a science of reasoning can be based: search strategies which systematically examine alternative steps and choose the right ones, means-end analysis, etc. They examined heuristic methods which employ rules of thumb which are *usually* successful, as against algorithmic methods which are guaranteed to work in every case. The focus was on general problem solving techniques, and not so much on problem-specific knowledge. The belief was that once we find good principles on which reasoning behaviour can be founded, we could just throw in relevant knowledge and start solving real problems.

Progress in the field had been slow in the sixties and seventies. General problem solvers have not been very successful. While Newton's three laws of motion go a long way in explaining the material world, there is no comparable power in any three laws of thought! Gradually a view is gaining ground that reasoning is a very complex phenomenon and we may need thousands of *rules,* and not just three or four laws, to explain the nature of reasoning.

1.5 'Search' as a General Problem Solving Technique

Certain classes of problems can be tackled reasonably well using a good search procedure. Consider, for instance, the problem of finding a path from A to B in the maze shown in Figure 1.2.

Assume that a robot, controlled by an AI program, has to navigate through this maze. Further, assume that the robot can provide the program only a little information at a time. It can be pointing east, west, north or south at a given time, and it can answer the question:

Are you blocked?

with a yes or now, indicating if it can move one square forward at that moment.

By making the program count the turns it orders the robot to make, and the steps that the robot is asked to take in each direction, one can ensure

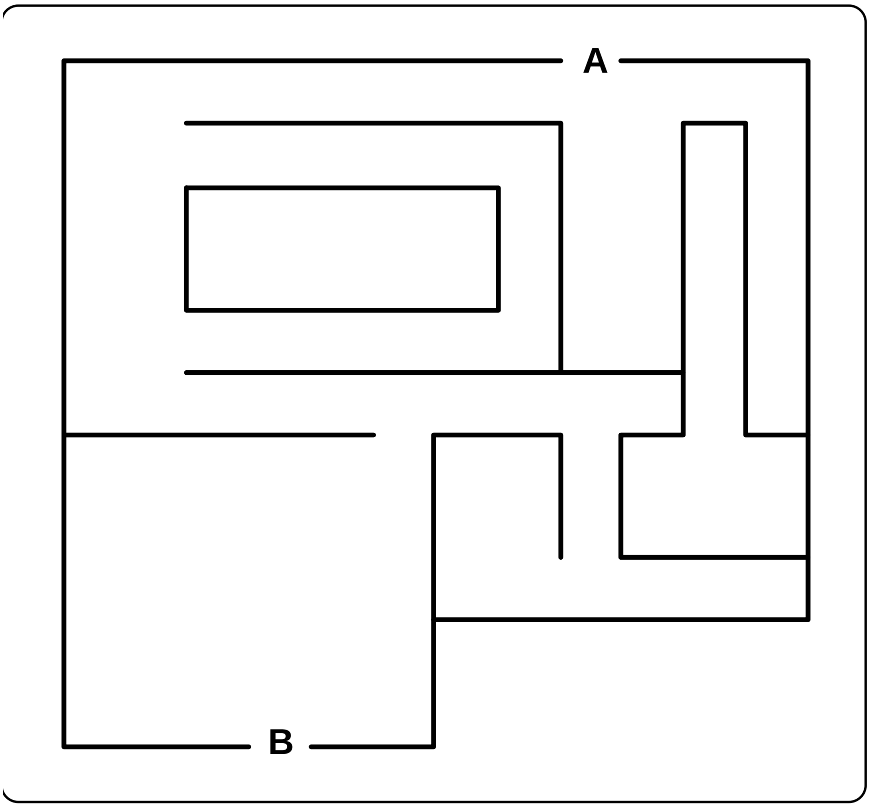

Figure 1.2: Finding a Path through the Maze

that the control program keeps track of the current position and orientation of the robot. The control program can also make sure that the robot does not run around in circular paths, which would otherwise trap it. If we avoid such “loops”, the maze becomes a tree-structured set of paths as shown in Figures 1.3 and 1.4.

It is easy to search a tree thoroughly. The tree in Figure 1.4 has been numbered in a systematic manner. A number has been given to each branch point, also called a non-terminal node, and to each end-point, called a terminal-node. The numbering starts at a designated node, numbered 1. All nodes reachable immediately from 1 are then numbered consecutively, 2, 3 and 4. Then we shift attention to node 2 and number the nodes directly reachable from 2 in one step. This numbering indicates an order in which we can search for a way out of the maze. A search in this order is called a breadth-first search. Going in the order 2, 3, 4 is searching breadth-wise.

It is also possible to search through the tree in a different order, named the depth-first order. A depth-first search of the tree in Figure 1.4 would follow the sequence:

1, 2, 5, 7, 9, 10, 11, 8, 6, 3, 4

Having reached the node 2, the depth-first technique explores all possible ways of reaching the exit through 2 and its sub-nodes, before trying paths through 3 and 4.

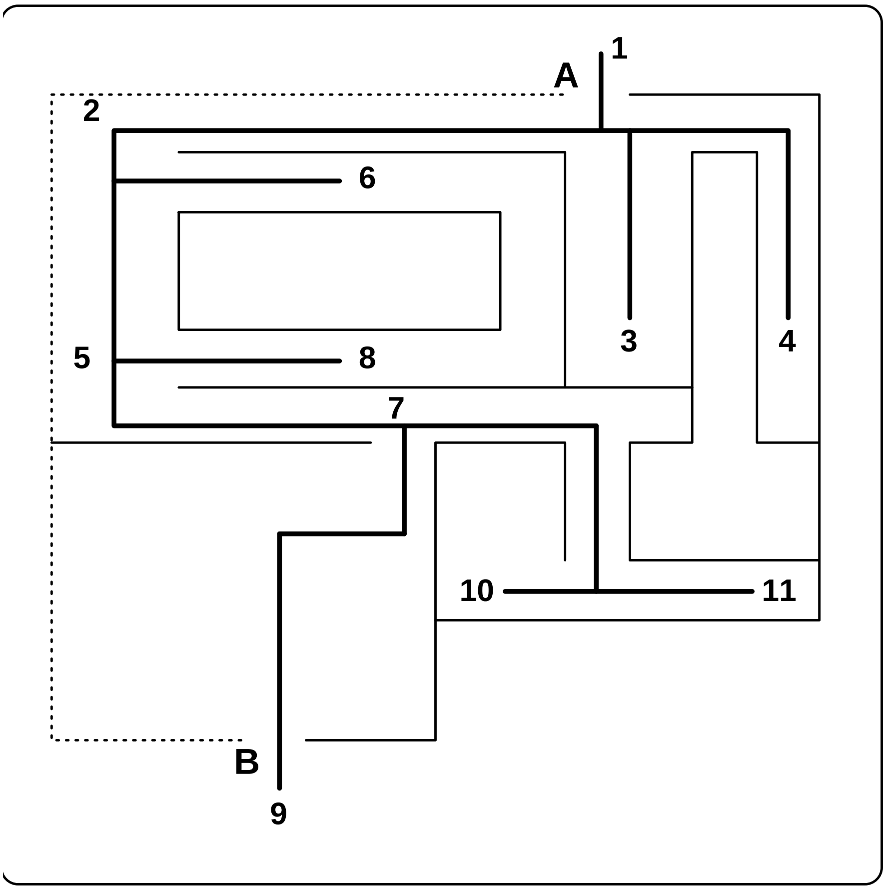

Figure 1.3: The Paths in the Maze after Loops are Removed. (The dotted lines show the original maze.)

1.6 Heuristic Search

The concept of tree search has been very effective in certain AI applications. For instance, modern chess playing programs use tree search to find out an appropriate response to a situation in the game, then to anticipate a possible response from the opponent, to identify a suitable response-to-the-response, and so on. By "looking ahead" several moves like this, a program imitates an interesting aspect of human chess playing behaviour. However, this imitation is less than perfect. A good human chess player does not "blindly" search all over the tree of possibilities. He makes guesses as to which paths are more likely to lead to the goal, and focuses more attention on selected parts of the tree. He also identifies parts of the tree that are clearly inferior to other parts he has examined, and abandons them quickly.

A variety of techniques are available for giving an AI program good search capabilities. These include

- best-first search, employing estimates of the value of the best possible next move; this uses a combination of breadth-first and depth-first views

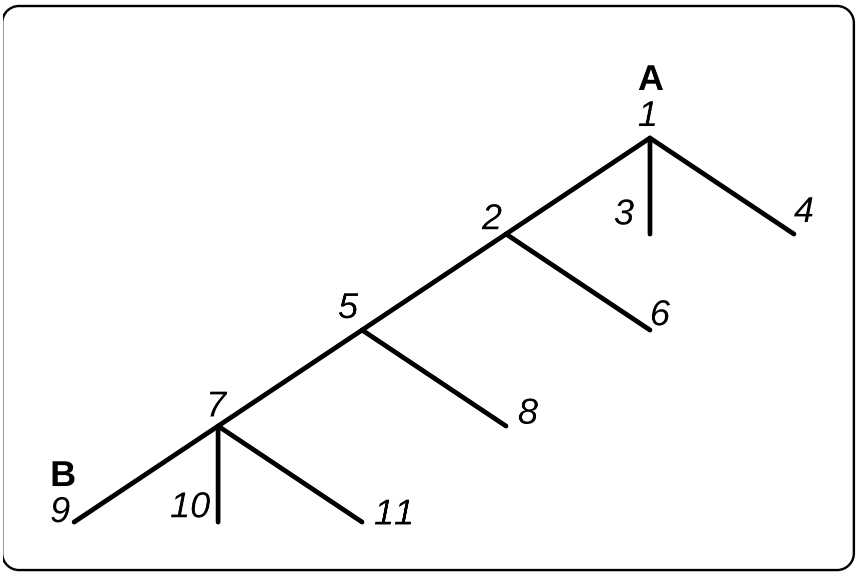

Figure 1.4: Basic Structure of the Tree in the Previous Figure

- the alpha-beta cut-off, employed in a competitive situation such as chess, for quickly identifying unattractive sub-trees.

We will not describe these techniques used in tree search in detail here. The focus in expert systems is, usually, on the use of “expert knowledge”, that is knowledge specific to a discipline. For instance, in chess, expert knowledge can tell you that a pair of bishops, in general, is worth more than a bishop and a knight. Individual value counts for the bishop and knight can both be 3, but expert knowledge can tell you that two bishops, in general, have better synergy than a bishop and a knight.

The focus on expert knowledge has, usually, distinguished “expert systems” from classical AI programs which use sophisticated tree search techniques. The reader may consult [Rich and Knight, 1991], [Charniak and McDermott, 1985] or [Nilsson, 1980] for more information on tree search.

1.7 The Resurgence of AI

Despite relatively slow progress in AI in the sixties and seventies, there were significant steps forward that led to substantial investments being made on AI in the eighties. The US Department of Defence listed AI as an essential technology and set up a strategic computing programme.

The Japanese launched a massive Fifth Generation Computer Systems project aimed at an entirely new generation of computers. Many of the goals of this project had strong AI components. The United Kingdom set up the Alvey Programme which made significant investments on “intelligent knowledge based systems”. This was quite a reversal of the earlier UK view which did not consider AI promising. The notorious Lighthill Report [Lighthill, 1973] had discussed the question of AI's promise as an area of investigation, and had made a very pessimistic judgement about the future of research in AI. Therefore, the willingness of the Alvey Programme to spend money on AI was a refreshing change.

The Microelectronics and Computer Technology Consortium (MCC) set up by a group of US companies recognised AI as an important area of work and launched significant efforts to solve major problems. The European Economic Community's ESPRIT Programme funded a number of AI activities. The Government of India and the United Nations Development Programme have initiated and supported the Knowledge Based Computer Systems Project being carried out at a number of leading institutions in India.

All this has led to a new phase in the field of AI research.

1.8 Limitations of Algorithmic Approaches

An algorithm is a computational procedure which, over a defined domain of problems produces solutions guaranteed to be accurate, using finite computing resources. The algorithm will complete its task in finite time. See Figure 1.5.

Humans solve a whole lot of problems in non-algorithmic ways. There is usually no algorithm which can do what they are doing. For instance, a medical doctor does not diagnose diseases using algorithms. Scientists do not use algorithms in making new discoveries, except as tools. The overall act of discovery is far richer and more complex than any algorithm. It is also notoriously uncertain in its outcome. A thousand scientists may work on a problem over their whole life-time. There are no guarantees of a discovery.

What do we have besides an algorithm, then? We frequently use *search*, trying out a lot of possibilities to find an answer. For problems of a small size, simple search works quite well. But if the problem involves a large number of variables, say, of the order of a hundred, brute-force search is not a practical solution.

For instance, consider the bin-packing problem, described in Figure 1.6.

For large values of k (say k > 100), there is no algorithm to solve the bin-packing problem in practical terms. No computer on earth is fast enough to run an algorithm for solving such large problems in times proportionate to a human life-time.

However, we should note that humans regularly solve problems similar to the bin-packing problem. Every professional who handles the layout of news items in a newspaper does it. Every mason tiling a floor using stone slabs does it quite well.

Bin-packing involving a hundred parcels can pose a computational task involving 2^{100} steps. Why? Because you can either choose to select or reject a parcel when you are selecting items to fill the bin. There are four options you can use in handling two parcels. There are eight ways of handling three parcels, and so on. Hence, as the number of parcels increases linearly, the complexity of the computation involved increases exponentially. Adding one more parcel to the initial lot doubles the time taken to solve the prob-

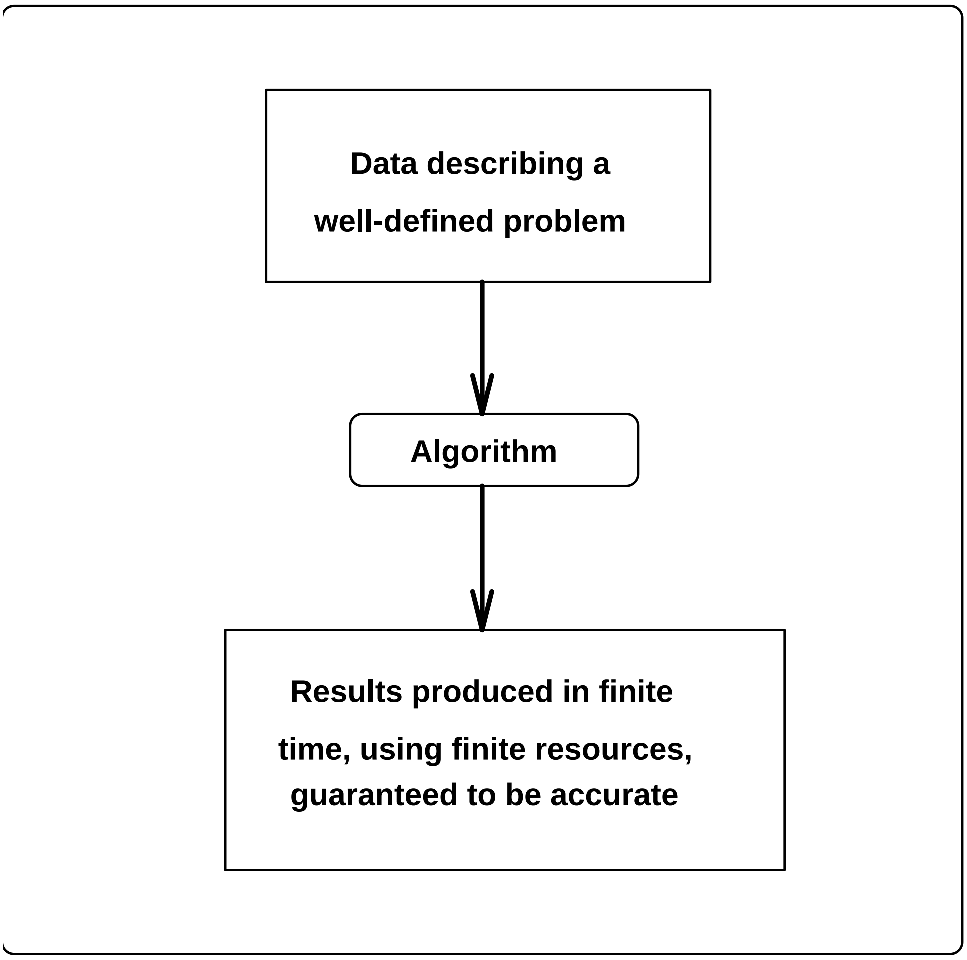

Figure 1.5: Algorithms

lem using simple search. There are no computers on earth that can do 2^{100} computations in practical time.

Hence, we go back to people and see how they handle problems such as this. One important, and early, finding of AI research was that people use *heuristics* (i.e., rules of thumb) in problem solving — ideas that usually work but are not guaranteed to work every time. Figure 1.7 depicts the difference between algorithms and heuristics.

For instance, you might find in bin-packing, a simple, but effective rule: pick up the biggest possible parcel from the lot; then choose the next biggest possible. If you proceed in this fashion, you can complete the task pretty fast. You may not get the absolutely minimal set required for the purpose, but you usually come pretty close. Is this cheating? After all, the problem was defined as that of finding a minimal set. What does it mean to come close to this solution? Are we solving a mathematical problem or not?

Heuristics are particularly valuable in handling practical problems we encounter every day. Bin-packing is not really a mathematical problem for the person handling newspaper layout. It is a very practical problem for him.

- **Task:** To select a set of parcels out of a given lot, to be packed in a container, so that the total weight of the selected parcels, T units, is maximised, subject to a weight limit of L units.
- In other words, to select a subset of parcels, numbered 1 to k, for some k, to maximise T, where
- T = $W_1 + W_2 + W_3 + ... + W_k$, and
- T $\leq$ L

  where W_i is the weight of the ith parcel.

Figure 1.6: A Version of the Bin-Packing Problem

Coming close is good enough for him. For instance, if an item of news almost fits in a particular place, but not completely, he will shorten that news item! Sometimes, he will cut off a paragraph, making sure that no significant information is lost.

Cutting the Gordian knot, that is what AI is all about! Making computer programs use short cuts and other ingredients of intelligence as human beings do.

Incidentally, the simple technique described above for solving the bin-packing problem is called the *biggest-first* heuristic.

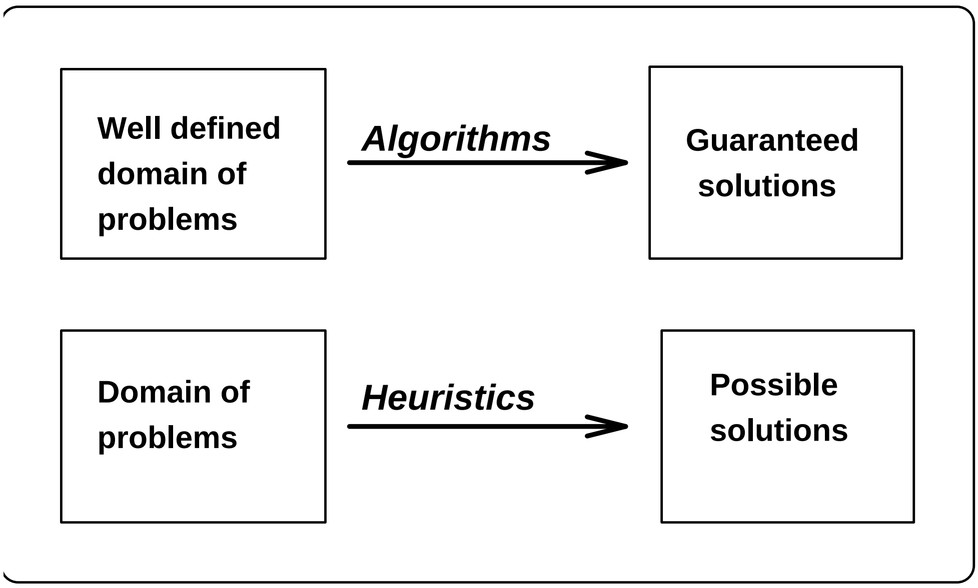

Figure 1.7: Algorithms vs Heuristics

1.9 Using Knowledge

As we noted in Section 1.4 the early work in AI concentrated a bit too much on search and the use of heuristics of a very general-purpose nature for guiding and limiting the search. A few examples can be found in [Feigenbaum and Feldman, 1963]. Chess can be played using the rules of the game, and a search strategy guided by a few heuristics. This was tried, but did not go far enough. Compare this with the strategies used by a human player, such as those listed in Figure 1.8.

- Use standard openings
- Dominate the centre
- Prefer to retain a pair of bishops rather than a bishop and a knight
- Other things being equal, prefer moves which unblock your pieces
- Use principles of standard end-play

Figure 1.8: Examples of Expert Knowledge in Chess

You can see the great emphasis placed on *knowledge*, not sheer search, not merely general purpose heuristics. The knowledge that a pair of bishops is better than a bishop and a knight, is clearly very specific "chess knowledge". It is such knowledge that separates experts in chess from the others! Once the importance of *domain-specific knowledge* was understood, research workers started building systems which exploited some computer representation (or "internal representation") of that knowledge to carry out complex tasks. One of the earliest successes was that of Feigenbaum's group at Stanford [Buchanan and Feigenbaum, 1978]. They reported a new type of program, named DENDRAL. It uses some of the knowledge a chemist uses, to find out what a chemical substance is, on the basis of measurements from a mass-spectrograph. The problem is akin to that of solving a chemical jig-saw puzzle. If you know the kind of things you should see when you complete your puzzle, it is a lot easier to solve it.

This change of direction led to a major paradigm shift in creating AI applications. Some of the features of this shift, which marked the birth of expert systems, are shown in Figure 1.9.

1.10 Expert Systems

... an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant expertise — Feigenbaum

There is a fair degree of agreement now about the use of the phrase *Expert System*. Typically, we demand that an Expert System (ES) should solve

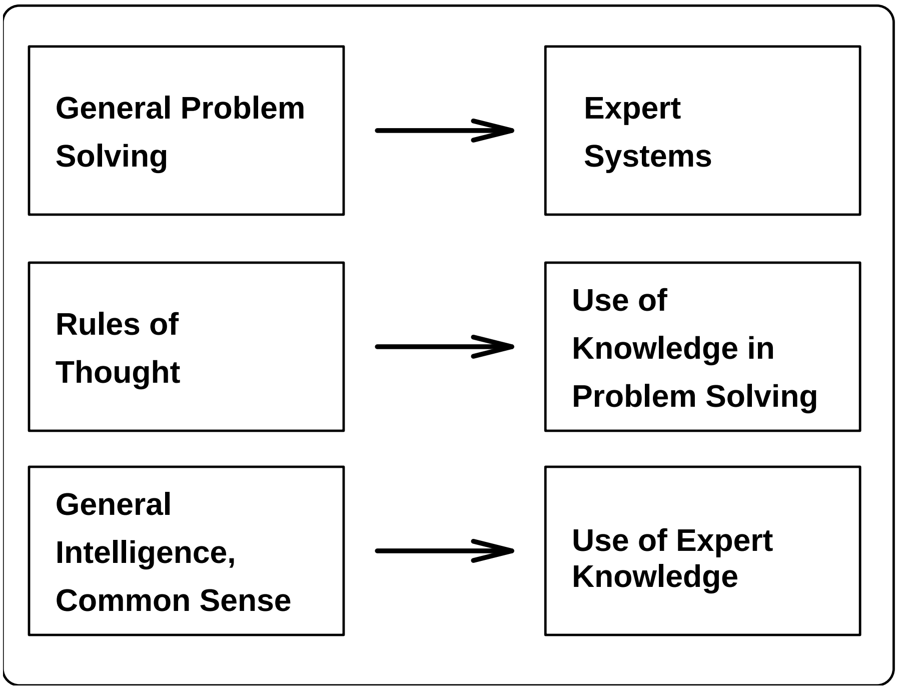

Figure 1.9: The Paradigm Shift in AI Applications

very difficult problems as well or better than human experts. They do not make significant use of algorithms in their main task. They use what are essentially rules of thumb, usually in a very *domain specific* form. They process information in symbolic form, and cope with errors in data and with imperfect rules of reasoning. They can simultaneously sustain conflicting hypotheses.

Another important feature of ESs is that their reasoning can be followed by a human being. In fact you can usually demand that an ES explains how it arrived at a particular conclusion. It will offer you an understandable explanation (more on this in Chapter 6).

For instance, if you ask an ES how it decided that a patient had paranoia, it might cite a few rules such as "belief that one is being persecuted could be a sign of paranoia", and "delusions of grandeur could be a sign of paranoia".

Also, in the middle of an interaction, when an ES is asking you for some information, you can demand to know why you are being asked that question. The system will offer you a reasonably good indication of its purpose. Suppose it was asking you:

> *Do you sometimes spend a whole day idle, because you do not find anything worthwhile to do?*

You could ask *Why are you asking me that?*, and you might get an answer such as this:

Spending a whole day idle because there is nothing worthwhile to do could be a sign of depression, and I wanted to see if you showed any signs of depression.

Later on, you will see the format in which these questions are normally asked and answered. The examples above might have made ESs look very human-like because the questions and answers were phrased in a style humans can understand. In practice, ESs use *canned* sentences and sound less human. However, these examples do illustrate the fact that ESs do not use some abstruse numerical reasoning. They follow human-like rules of thought. They can answer *why* and *how* questions reasonably well.

The researchers working with medical diagnosis by computer made major contributions to the development of ESs. A particularly noteworthy contribution was that of Buchanan and Shortliffe [Buchanan and Shortliffe, 1984]. The key issue that they had to face was that of using information that was not simply "true-or-false", but had different levels of certainty associated with it. For instance, we could characterise a patient's status in the following terms:

- chest-pain: definite
- obese: almost
- sedentary-job: yes
- hereditary-predisposition-to-heart-problem: high

Shortliffe came up with a simple calculus of *certainty factors* which enabled one to use information such as this and to combine evidence from many individual bits of information. MYCIN [Buchanan and Shortliffe, 1984] was a pioneering ES which demonstrated the value of this technique of reasoning under uncertainty. MYCIN deals with the identification and treatment of blood infections. It handled diagnosis and suggested appropriate medication and dosage. It used a variety of information, including patient's complaints and pathological laboratory reports. Even healthy people harbour bacteria, and therefore pathological tests could be occasionally misleading. MYCIN took such possibly misleading evidence and managed to use it intelligently. It also had information reflecting the fact that a bacterium normally susceptible to a given antibiotic may turn out, in some individuals, to be resistant to that antibiotic.

The very high quality performance of MYCIN, in the face of such uncertainty, went a long way to convince the world of the power of the ES paradigm.

The whole issue of reasoning under uncertainty is discussed in detail in Chapter 5.

1.11 When to Invest in Creating an ES

ESs have been called white-collar robots. They are not endowed with anything like common-sense. In this sense, they are like robots, a bit mechanical, simple-minded and domain-specific. But they use domain-specific knowledge gleaned (by their creators) from human experts to handle a variety of situations reasonably competently. They have shown their practicality in dealing with a variety of frequently occurring situations.

For instance, consider a situation which needs scarce expertise, say in predicting side-effects of drug combinations. This could be coupled with heavy work-load, as in for instance, an out-patient department of a hospital which gives out thousands of prescriptions per day. The situation could be worse if work is in shifts, which usually makes it difficult to get an adequate number of human experts to stay on. Another major factor may be that relevant information is available on a computer. For instance, in this case, patient records might already be on the computer indicating what regular medication is being taken by which patient, what are the patient's allergies, etc. Systematically looking all this up in the context of a new problem and finding possible side-effects would be ideal for the tireless efforts of an ES.

Also, consider the fact that it takes a long time to train humans to be knowledgeable about all side-effects of drugs and that such trained human experts are difficult to get. Note that speedy decisions have to be made in handling these problems. Again an ES meets the needs of this type of application.

Some of the characteristics of an area which make it amenable to the ES approach are listed in Figure 1.10.

- It is an important, high-value problem area
- An expert is available to share his knowledge of the field
- The problem can be solved by talking to an expert on the phone
- An expert can solve it in a few minutes
- There is no argument about what constitutes a good solution
- People who solve this problem build up their skills gradually
- There is a book/manual which tells you how to do this work

Figure 1.10: Some Features of Areas Suitable for Expert Systems

1.12 Structure of a Typical Expert System

Figure 1.11 shows the structure of a typical expert system and its component parts. Running on an interactive computer, typically a personal computer (PC) or a work-station, is a piece of software called an *inference engine.* Associated with the *inference engine* is a *knowledge base*, containing rules of inference and related factual information, about a particular domain. Along with the inference engine, there is need for a good interface for interaction with an expert who creates knowledge bases and with the naive end-user of the ES. This interface should offer good support for man-machine interaction, making it easy to put in rules and to edit them. The whole of the software that enables you to build and use ESs is called an *expert system shell* . Shells are usually quite generic. You can easily develop diverse ESs using a shell as a base. The domain-specific knowledge is in the knowledge base, which is usually created by a *knowledge engineer* (KE) collaborating with a *domain expert.*

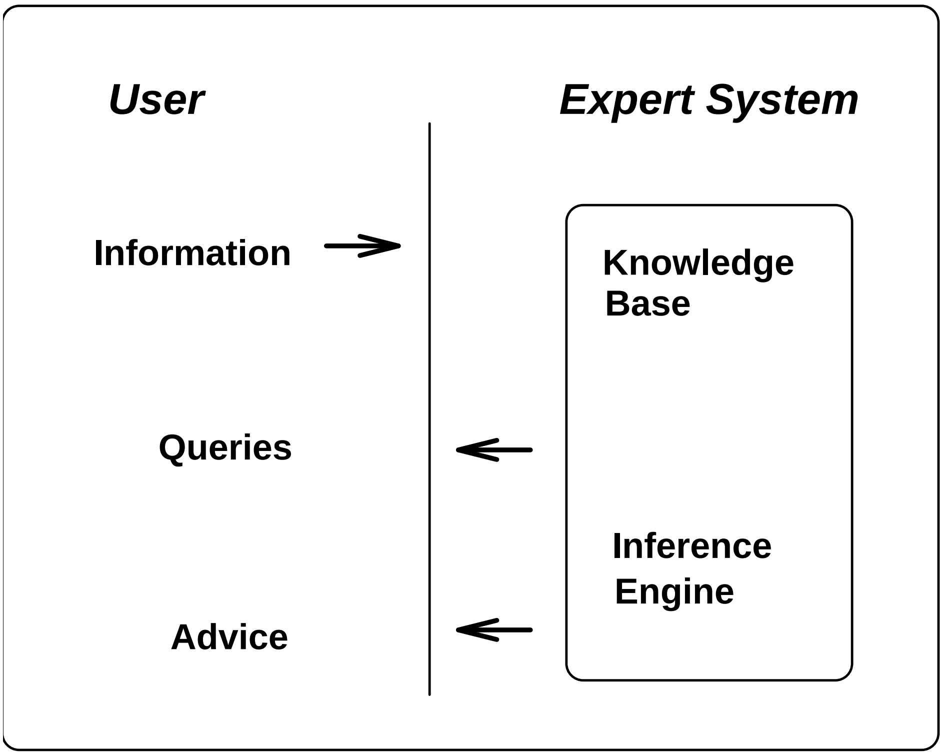

Figure 1.11: Structure of a Typical Expert System

Typically, a knowledge base consists of hundreds of rules of the form shown in Figure 1.12. Items such as *age* and *nails-blue* are called parameters. Some of them, such as *age*, take a numerical value, while others, such as *nails-blue*, take a boolean value: true or false. The inference engine applies (or *fires*) the rules and computes values of parameters dependent on the rules. One of the major tasks of the inference engine is to schedule the rules, deciding in what order they should be fired. The inference engine also handles *certainty factors (CF)* such as the 0.80 shown in the example of a rule. This means that the conclusion is true with certainty 0.80 if all the conditions are true with absolute certainty.

1.13 Knowledge Acquisition

A person knowledgeable about expert systems, and having experience in using expert systems is required to function as a knowledge engineer (KE). In addition to the technical knowledge, he has to have some creative ability. For, the creation of an ES is not the blind application of rote knowledge. Some of the creative urge of an architect is necessary in a knowledge engineer. The rules are not a mass of detail. They usually show structure at different

Rule sd4:
If age < 40
and nails-blue = true
and cardiac-symptoms-present = true
Then suspected-disease = congenital-heart-disease **with CF =** 0.80

```
rule(sd4, suspected-disease, congenital-heart-disease, 0.80) :-
        lt(age, 40),
        is(nails-blue, true),
        is(cardiac-symptoms-present, true).
```

Figure 1.12: Example of a Rule in Two Equivalent Forms

levels of organisation. For instance, there may be different stages in making a medical decision: noting the symptoms, making a hypothesis, verifying the hypothesis, ruling out rare exceptions, etc. The knowledge base will show these stages separately. There will be groups of rules dealing with different issues. A good knowledge engineer has experience in structuring knowledge bases in the light of all this.

The KE works with a domain expert, for example, a physician when dealing with medical diagnosis. The domain expert provides the knowledge which is to be represented in the knowledge base. He usually explains the structure of reasoning in the domain. By watching him make decisions, and having him think aloud about them, one gets the basic feel for the rules that are required.

We should also not under-estimate the value of published information in creating a knowledge base. Textbooks, reference books, manuals, etc., are gold mines of information. A creative KE can gain a lot of the relevant information from these sources.

Published sources of information very often offer statistical information, unlike human experts who are often not knowledgeable about statistical information in their domains. Log books, serial case studies and such similar information sources yield a lot of statistical information. It is rare for a KE to use statistical information formally. But he gets a good idea of what certainty factors should be used by finding out what is frequent, what is common, what is rare and so on.

After the KE creates a set of rules using the domain expert's knowledge, he can run the ES and compare its behaviour with that of the expert. By critically examining its behaviour and finding out what is right and what is wrong with it, one can modify the rules to improve the system's performance. Finally, after considerable observation of the system, one may consider it "validated".

Issues in knowledge engineering are discussed in detail in Chapter 8.

1.14 The Nature of Expertise

Human knowledge is often inexact and incomplete. It takes a long time for a human apprentice in a trade to become an expert. He acquires knowledge step-by-step. Thus, novices become experts gradually. All this points to the fact that a KE cannot do a two-week job of creating a significant expert system. ESs tend to evolve over a period of time. A demonstration ES can be created for many domains in one day. But it may take months to evaluate the system and to refine it, to a level of reasonable performance.

Frequently, the experience of teaching new-comers to the area is helpful in creating an expert system. A KE's role is very similar to that of a teacher.

1.15 Conclusion

In this chapter, we started with a brief look at the history of artificial intelligence and of expert systems. We discussed the shift of focus from general-purpose rules of thought to knowledge based rules that experts seem to use. We looked into the nature of expertise and the structure of expert systems. Later chapters will complement the coverage here. In particular, we should note the discussion on limitations of Rule Based Systems in Chapter 2, and the discussion of anticipated developments in Chapter 10.

Summary

- One of the goals of Artificial Intelligence is to create computer programs which exhibit some features of human intelligence.
- The Turing Test is a test to determine whether a computer system is *intelligent* or not.
- Early attempts in creating AI programs was aimed at creating general purpose problem solvers using some form of *search*.
- The most common search procedures are breadth-first, depth-first and best-first (search using heuristics).
- The failure to create successful general-purpose problem solvers led to the development of the field of Expert Systems. Expert Systems use domain specific knowledge to solve problems.
- Certain classes of problems such as the bin-packing problem cannot be solved by algorithmic methods and are often solved using heuristics.
- Heuristics often represent human expertise. They are not guaranteed to be correct all the time, but work much of the time.
- In domains such as chess, domain-specific knowledge plays a very important role.

- An expert system is an intelligent program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant expertise.
- An expert system contains a knowledge base and an inference engine.
- Domains should first be evaluated to determine whether they are suitable for expert system application.
- Knowledge Acquisition is the process of acquiring knowledge from the domain expert, so that it can be put in a form suitable for direct use by an expert system.

Exercises

1. What are the main differences between conventional programs and expert systems?
2. Select an area, not mentioned in this chapter, that is suitable for work using the ES approach. List all the knowledge required to handle a class of problems in this area.

   Try and solve a few problems using only the listed knowledge. If that does not work, add a rule or two to the list of rules you are using, and go on. It does not matter if the rules are not written in any particular form. As long as it is clear what a rule says, you can add it to the list.
3. Consider the areas given below:

   (a) To look at bad photographs and tell a person what could have gone wrong in taking those pictures.

   (b) To listen to a person on the telephone about car trouble and advise him what to do.

   (c) To listen to a person complaining about his telephone and to locate the problem.

   (d) To ask questions about a patient's symptoms and identify a common disease causing them.

   Try using the approach mentioned in the previous exercise for each of these areas and see how hard or easy it is to identify the relevant knowledge and to put it in the form of rules.
4. Examine the Turing test example given in Figure 1.1. What are the pieces of knowledge that the machine exhibited there? Can we put them down in the form of rules? Are there similar items of knowledge not shown in the test in the figure? Can you list some of them? Is it possible for an ES to pass the Turing Test, even using a thousand rules to capture the relevant knowledge? Or, will the system need commonsense, going beyond current ES technology?

5. In Figure 1.8, we had given examples of types of knowledge which would be required to play chess. Can you think of other types of knowledge which would be useful for a chess playing program? What would your estimate be of the number of *rules* a machine would need to become a grandmaster?

6. For each of the following tasks, decide whether the expert systems approach is applicable. Justify your answers.

   (a) Solving a set of simultaneous equations

   (b) Computing the income tax for a person, given the required information

   (c) Diagnosing a fault in a television set

   (d) Predicting the lifespan of a person

   (e) Measuring the IQ (intelligence quotient) of a person

7. The 8-puzzle is a 3 x 3 square tray in which are placed 8 square tiles as shown in the figure below:

1	2	3
4	5	6
7	8	

   The remaining ninth square is uncovered. Each tile has a number on it. A tile that is adjacent to the blank space can be moved into that space. A game consists of a starting position and a specified goal position. The goal is to transform the starting position into the goal position by moving the tiles around. Think of a possible representation for this problem. Outline a solution strategy to find out the sequence of moves to reach the goal position from a given starting position.

8. For each of the following domains, list out the sources of information you would use for acquiring knowledge about the domain to put in an expert system.

   (a) Troubleshooting cars

   (b) Designing a layout for a house

   (c) Deciding whether someone is eligible for an insurance policy and determining the premium for the policy if he is eligible

   (d) Diagnosing common diseases

2. Rule Based Systems

2.1 Introduction

If-then rules are one of the most common forms of knowledge representation used in expert systems. Systems employing such rules as the major representation paradigm are called rule based systems. Some people refer to them as production systems. There are some differences between rule based systems and production systems, but we will ignore these differences and use the terms interchangeably.

One of the first popular computational uses of rule based systems was the work by Newell and Simon on the General Problem Solver [Newell and Simon, 1972]. In this work, production rules were used to model human problem solving behaviour. However the mathematical model of production systems was used much earlier by Post in the domain of symbolic logic [Post, 1943].

Work on rule based systems has been motivated by two different objectives. One of these is Psychological Modelling. The aim here is to try to create programs that embody a theory of human performance of simple tasks and reproduce human behaviour. The theory should *explain* all aspects of *human* performance. There are a number of theories which use rules as their basis and try to explain human behaviour. The most common are SOAR [Rosenbloom et al, 1991] and ACT$*$ [Anderson, 1983].

The other objective, the one which we will be concentrating on in this chapter, aims at creating expert systems which exhibit competent problem solving behaviour in some domain. However, the exact problem solving process used by the system need not be similar to that used by humans.

2.2 Components of a Rule Based System

A typical rule based system consists of three components (Figure 2.1). They are:

- the working memory,
- the rule base, and
- the inference engine

The rule base and the working memory are the data structures which the system uses and the inference engine is the basic program which is used. The advantage of this framework is that there is a clear separation between the data (the knowledge about the domain) and the control (how the knowledge is to be used).

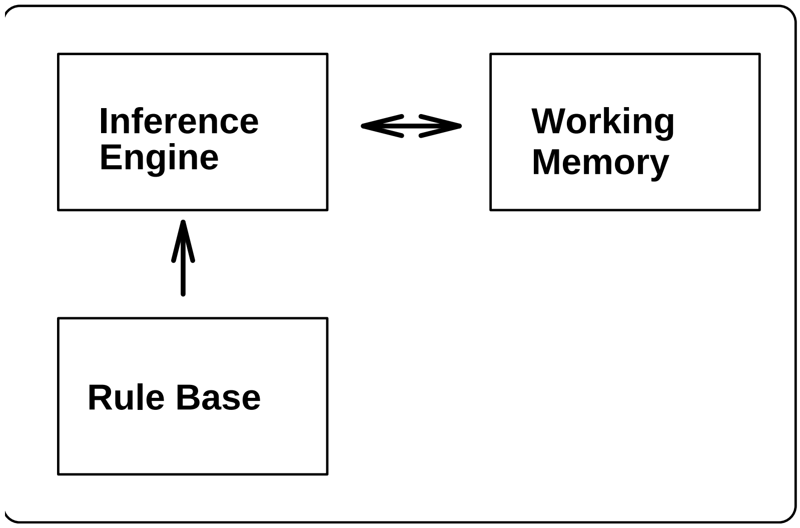

Figure 2.1: Components of Rule Based Systems

2.2.1 Working Memory

The working memory (WM) represents the set of facts known about the domain. The elements reflect the current state of the world. In an expert system, the WM typically contains information about the particular instance of the problem being addressed. For example, in a medical expert system, the WM could contain the details of a particular patient being diagnosed.

The working memory is the storage medium in a rule based system and helps the system *focus* its problem solving. It is also the means by which rules *communicate* with one another.

The actual data represented in the working memory depends on the type of application. The initial working memory, for instance, can contain *a priori* information known to the system. The inference engine uses this information in conjunction with the rules in the rule base to derive additional information about the problem being solved.

In this chapter we will use working memory elements of the form:

colour(car5, black)	to represent the fact that the colour of car *car5* is black
father(mohan, vivek)	to represent the fact that Mohan is the father of Vivek

This is only one way of representing information in the WM. There could be other ways depending on the application.

2.2.2 Rule Base

The rule base (also called the knowledge base) is the set of rules which represents the knowledge about the domain. The general form of a rule is:

If cond1
and cond2
and cond3
...
then action1, action2, ...

The conditions *cond1, cond2, cond3,* etc. (also known as antecedents) are evaluated based on what is currently known about the problem being solved (i.e., the contents of the working memory). Some systems would allow disjunctions in the antecedents. For example, rules like the following would be allowed.

If cond1
and cond2
or cond3
...
then action1, action2, ...

Such rules are interpreted to mean that if the antecedents of the rule together evaluate to true (i.e., if the boolean combination of the conditions is true), the actions in the consequents (i.e., action1, action2, etc.) can be executed.

Each antecedent of a rule typically checks if the particular problem instance satisfies some condition. For example, an antecedent in a rule in a medical expert system could be: *the patient has previously undergone heart surgery.*

The complexity of antecedents can vary a lot depending on the type of language used. For instance, in some languages, one could have antecedents such as: *the person's age is between 10 and 30.*

The consequents of a rule typically alter the WM, to incorporate the information obtained by application of the rule. This could mean adding more elements to the WM, modifying an existing WM element or even deleting WM elements. They could also include actions such as reading input from a user, printing messages, accessing files, etc. When the consequents of a rule are executed, the rule is said to have been *fired*.

Sometimes the knowledge which is expressed in the form of rules is not known with certainty. In such cases, typically, a degree of certainty is attached to the rules. These degrees of certainty are called *certainty factors* and are described in Chapter 5.

In this chapter we will consider rules with only one consequent and one or more antecedents which are combined with the operator **and**. We will use a representation of the form:

ruleid: **If** antecedent1 **and** antecedent2 .... **then** consequent

For instance, to represent the rule that all birds can fly, we use:

f1: **If** bird(X) **then** can_fly(X)

This representation, though simple, is often sufficient. For instance, if you want to represent the knowledge that either a bird *or* [1] a plane can fly, you can do this by using two rules f1 and f2 as follows:

f1: **If** bird(X) **then** can_fly(X)

f2: **If** plane(X) **then** can_fly(X)

Therefore the disjunction (ORing) of a set of antecedents can be achieved by having different rules with the same consequent.

Similarly, if multiple consequents follow from the conjunction (ANDing) of a set of antecedents, this knowledge can be expressed in the form of a set of rules with one consequent each. Each rule in this set will have the same set of antecedents.

2.2.3 Inference Engine

The inference engine tries to derive new information about a given problem using the rules in the rule base and the situation-specific knowledge in the WM.

At this stage, we need to understand the notion of an instantiation. Consider an example to illustrate the idea of an instantiation. Suppose the working memory contains the elements:

bird(crow)
bird(eagle)
aircraft(helicopter)

With these working memory elements rule f1 (given in the previous section) can fire. However the antecedent of f1 actually matches [2] two working memory elements. The instantiated antecedents and the corresponding consequents are given below:

f1:	bird(crow)	can_fly(crow)
f1:	bird(eagle)	can_fly(eagle)

Each of these matches (rule id and matching antecedent(s)) is called an *instantiation* of the rule f1.

Given the contents of the working memory, the inference engine determines the set of rules which can be *fired*. These are the rules for which the antecedents are satisfied. The set of rules which can be fired is called the *conflict set*. Out of the rules in the conflict set, the inference engine selects one rule based on some predefined criteria. This process is called *conflict resolution* and is described in the next section. For example, a simple conflict resolution criterion could be to select the first rule in the conflict set. The creation of the conflict set, selection of a rule to fire and the firing of the rule are together called the *recognise-act cycle* of the inference engine.

The action corresponding to a rule could be to add an element to the working memory, delete an element from the working memory or change an existing working memory element. It could also include some actions which do not affect the working memory, for instance printing out the value of a working

[1] This is an inclusive *or*.

[2] An antecedent matches a working memory element, if the antecedent is identical to the working memory element or if by binding the variables in the antecedent to suitable constants, the antecedent and the working memory element become identical.

memory element.

Since firing a rule may modify the working memory, in the next cycle, some instantiations present earlier may no longer be applicable; instead, some new rules may be applicable. The conflict set will therefore change and a different instantiation may be chosen in the next cycle.

This process is repeated till, in a given cycle, the conflict set is empty or an explicit *halt* action is executed, at which point the inference engine terminates.

2.3 Conflict Resolution

The strategy used for selecting one rule to fire from the conflict set is called the *conflict resolution strategy*. The behaviour of the system is dependent on the conflict resolution strategy or strategies used.

There are, a number of sophisticated conflict resolution strategies which are used by commercial systems. These include:

- Specificity
- Recency
- Refractoriness

Each of these strategies will be briefly described in the following subsections. Normally a single conflict resolution strategy is not adequate for an application, so systems have to often use a combination of strategies. For example the OPS5 system, which we will discuss in Section 2.7, uses a combination of all three of the strategies mentioned above in different ways [Cooper and Wogrin, 1988].

2.3.1 Specificity

Using this strategy, rules with more antecedents (conditions) are preferred to rules with less conditions. That is, more specific rules are selected in preference to general rules.

For example consider the two rules:

p1: **If** bird(X) **then** can_fly(X)

which represents the knowledge that **if** X is a bird **then** X can fly.

p2: **If** bird(X) **and** ostrich(X) **then not** can_fly(X)

which represents the knowledge that **if** X is a bird **and** X is an ostrich **then** X cannot fly.

Suppose the working memory contains the facts:

bird(b1)	b1 is a bird
ostrich(b1)	b1 is an ostrich

Then by binding the variable X in the antecedent bird(X) of p1 to *b1*, we find that the antecedent succeeds. Similarly, both the antecedents of rule p2 are true, if X is bound to *b1*.

Using specificity, rule p2 would be selected to fire, because its antecedents are a superset of the antecedents for p1. As can be seen, one use of specificity is to put knowledge about exceptions into the rule base. The general idea behind specificity is that the more specific rules are tailored to specific problems and hence should be given preference.

2.3.2 Recency

With this strategy, every element of the working memory is tagged with a number indicating how recent the data is. When a rule has to be selected from the conflict set, the rule with an instantiation which uses the most recent data is chosen. If the most recent data item is matched by more than one instantiation, the time of the second most recent data element for each of these rules is compared and so on.

The idea here is that a rule which uses more recent data is likely to be more relevant than one which uses older data. This criterion provides a simple attention focussing mechanism.

2.3.3 Refractoriness

Refractoriness prevents the same rule from applying again and again. If an instantiation has been applied in a cycle, it will not be allowed to fire again.

Refractoriness is important for two reasons. It prevents the system from going into a loop (i.e., repeated firing of the same rule with the same instantiation). It also improves the efficiency of the system by avoiding unnecessary matching.

2.4 An Example

To get a better feel of rule based systems, we will consider a simple example which deals with relationships between members of a family.

Consider the following rule, defining the relationship *mother*.

p1: **If** father(X,Y) **and** wife(Z,X) **then** mother(Z,Y)

which represents the knowledge that **If** X is the father of Y **and** Z is the wife of X **then** Z is the mother of Y.

We can add similar rules to represent other relationships. Some rules are given below:

p1: **If** father(X,Y) **and** wife(Z,X) **then** mother(Z,Y)
p2: **If** mother(X,Y) **and** husband(Z,X) **then** father(Z,Y)
p3: **If** wife(X,Y) **then** husband(Y,X)
p4: **If** husband(X,Y) **then** wife(Y,X)
p5: **If** father(X,Z) **and** mother(Y,Z) **then** husband(X,Y)
p6: **If** father(X,Z) **and** mother(Y,Z) **then** wife(Y,X)

Consider the following facts which comprise the working memory. Each element has a time tag which indicates the chronological order in which the element was added (more recent elements have higher time tags):

father(rama, mohan)	1
mother(alka, lata)	2
wife(lata, hari)	3
wife(alka, rama)	4
father(hari, uma)	5

The relationships between the different people given in the working memory is also pictorially depicted in Figure 2.2.

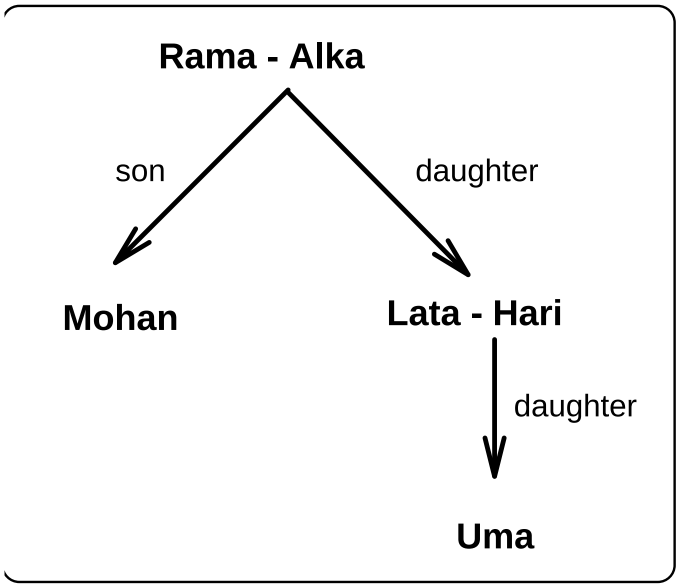

Figure 2.2: A Family Hierarchy

The conditions in rule p1 could match with the facts [3]:

father(hari, uma)
wife(lata, hari)

If p1 is chosen for firing with these matches, the fact *mother(lata, uma)* would be added to the working memory.

Now consider all the rules given earlier. The system will derive new information based on the initial contents of the working memory and the set of rules it has. The facts the other rules would match are:

p2: no facts matched
p3: wife(lata, hari) and wife(alka, rama)
p4: no facts matched
p5: no facts matched
p6: no facts matched

Assume that the conflict resolution strategy is a combination of three different strategies in the order given below:

- Refractoriness
- Recency

[3]We have considered the more recent working memory elements first.

- Specificity

Refractoriness is used to ensure that the same rule will not be fired more than once with the same instantiation. Recency is used to select between different instantiations of a single rule which match the working memory elements. Specificity is to ensure that more specific rules are fired first. If after the application of these three strategies, the conflict set still contains more than one rule, then the textual order of the rules in the rule base is used to resolve the conflict. The rule which comes first in textual order is selected to fire.

The exact sequence of rule firings under this conflict resolution strategy and the facts which are added to the working memory are given below. The system will keep on deriving new facts until the conflict set is empty. Rules which add working memory elements which are already present have been omitted from the list. Try simulating the inference engine to see how these new facts were derived (the facts are preceded by the name of the rules which were fired to derive them).

p1: mother(lata, uma)
p5: husband(hari, lata)
p1: mother(alka, mohan)
p5: husband(rama, alka)
p2: father(rama, lata)

2.5 Conventional Programs vs Rule Based Systems

From the example given above and our description of rule based systems, the difference between conventional programs and rule based systems can be summarized as follows.

- The major feature which differentiates a rule based system from a conventional program is its declarativeness. In a rule based system, the knowledge of the world is stated declaratively in the form of rules and facts. A control mechanism is used to infer new facts about the world. In a conventional program, such a clean separation does not exist.

- A rule based system can also be given a procedural interpretation. However, the nature of procedure invocation differs from that of conventional programs. The rules do not invoke other rules directly. Rules modify the contents of the working memory. This modification of the working memory causes other rules to become *fireable*. This is unlike procedure invocations in conventional programs.

- A rule based system exhibits a high degree of modularity compared to most conventional programs. Each rule represents an independent piece of knowledge. Therefore the addition of a rule to the rule base need not affect any of the other rules. In contrast, in conventional programs, the addition of a new procedure would involve changes in the existing code.

2.6 Forward and Backward Chaining

Inference engines in rule based systems can use different strategies to derive the goal (i.e., new fact), depending on the types of applications they are aimed at. The most common strategies are:

- Forward Chaining
- Backward Chaining

Systems can use either one of these strategies or a combination of both. Some systems allow users to specify which strategy they need for their application. These strategies are briefly described below.

2.6.1 Forward Chaining

The inference strategy we have been using so far in this chapter is called forward chaining.

Forward chaining is essentially data driven. That is, the system starts with the initial set of elements in the working memory and keeps on firing rules until there are no rules which can be applied or the goal has been reached. In effect, the system is moving *forward* from the current state to a goal state. In our *family* example given earlier this is the strategy we followed.

This strategy is suitable for applications in which the number of goal states is large. That is, there is no single, unique goal state. The goal state is required to satisfy some constraints.

The tasks of planning, design and synthesis are in general best suited for forward chaining.

2.6.2 Backward Chaining

Backward chaining is a goal driven strategy. Backward chaining involves decomposing a problem into sub-problems and solving each one of them. That is, the goal is reduced to sub-goals and each sub-goal is reduced further, and so on, until they are solvable directly.

In the context of rule based systems, backward chaining is achieved by the following algorithm:

1. Assume the goal is to infer a fact, say G. Set current goal to G.
2. Find the rules which can infer G. If there are no rules for G use step 4.
3. In each of the rules look at the sub-goals to be solved. Treat each sub-goal as a new goal and repeat steps 2 and 3 with the new goal as current goal. If all sub-goals in a rule succeed, the conclusion of the rule can be added to the WM.

4. If G matches a working memory element, it succeeds or else it fails[4].

Backward chaining is used when the number of possible initial states are large, compared to the number of goal states.

The tasks of classification and diagnosis are best suited for backward chaining.

2.7 The OPS Programming Language

OPS is a class of rule based languages [5] which are used for forward chaining applications. In this section we try to give you a feel for OPS, by describing how the example in Section 2.4 can be represented in OPS.

In OPS the working memory elements are instances of predefined classes. Each class has a name and a set of attributes. For instance, if you would like to define a class named `Father`, you would have a declaration similar to the following:

```
(literalize Father
     name    ; name of the person
     father  ; father's name
)
```

The class `Father` has attributes name and father. When an instance of this class is created for the working memory, these attributes take on values. Similar classes can be defined for `Mother`, `Husband` and `Wife`. (refer to Figure 2.3a) [6].

To represent that mohan's father is rama, we would have an instance of the class `Father` as follows:

```
(Father ^name mohan ^father rama)
```

The types of attributes and the values of attributes are distinguished by the fact that types of attributes are preceded by the character `^`.

In OPS, rules have an associated name, a set of antecedents and a set of consequents. For example the OPS rule equivalent to rule p1 in Section 2.4 would be:

```
(p p1
   (Father ^name <y> ^father <x>)
   (Wife ^name <x> ^wife <z>)
   -(Mother ^name <y> ^mother  <z>)
  -->
   (make Mother ^name <y> ^mother  <z>))
```

[4] In reality, if G is not a working memory element, one would typically ask the user for the value of G, before failing.

[5] We will use the term OPS to represent the whole class of languages from now onwards.

[6] A more general class *Person* with attributes, name, father, mother, husband and wife could have been created instead of having four different classes, but for reasons of simplicity we decided to have different classes.

The *p* after the bracket indicates that it is a (production) rule. The name of the rule is p1. Variables are indicated by enclosing them within $<$ and $>$. The rule can be interpreted as:

If X is the father of Y **and** Z is the wife of X **then** Z is the mother of Y.

Another way of interpreting this rule is:

If there exist working memory elements of the form `(Father ^name <y> ^father <x>)` and `(Wife ^name <x> ^wife <z>)`, and if the element `(Mother ^name <y> ^mother <z>)` does not exist (the $-$ before the antecedent denotes that the antecedent should be negated) then add a new element of the form `(Mother ^name <y> ^mother <z>)` to the working memory.

The *make* command in the consequent is used to create a new working memory element. So after the variables x, y and z are instantiated, a new element which is an instance of the class `Mother` is added to the working memory.

There is one special rule named *Initialize* in the rule base in Figure 2.3a. This rule initializes the contents of the working memory before the other rules are applied. This rule is fired if there is an instance of the class *Start* in the working memory. Therefore, after the rule base is loaded, if the working memory element of class Start is added (this can be done by giving the command (make Start)), then when the system is run, the rule Initialize will be first fired. This will lead to the creation of the other working memory elements, which will in turn lead to the firing of the other rules.

The special thing about the Initialize rule is that after it finds a working memory element which matches the antecedent (Start), it binds the number of that element to the variable *initialize*. At the end of the Initialize rule, this working memory element is removed from the working memory by using the *remove* command. The remove command can be used to delete a working memory element given the number of that working memory element. In this example, it is not strictly necessary to remove the instance of the *Start* element, but it is desirable to do so for efficiency reasons. Deletion of this rule, essentially makes sure that the Initialize rule does not enter the conflict set again.

The complete OPS program for our family example is given in Figures 2.3a and 2.3b. OPS by default uses a strategy named LEX for conflict resolution. This strategy applies refractoriness, recency and specificity in that order while doing conflict resolution. Running the OPS program would add the same working memory elements as in the example in Section 2.4 and would add them in the same order.

We have just scratched the surface of the OPS language. The language has a number of other features, like allowing disjunction of antecedents, allowing antecedents to be negated, allowing working memory elements to be modified, providing input-output support and a number of other functions. Interested readers can refer to [Cooper and Wogrin, 1988] for more details.

```
(literalize Start
        ; no attributes
)

(literalize Father
     name    ; name of the person
     father  ; father's name
)

(literalize Mother
     name    ; name of the person
     mother  ; mother's name
)

(literalize Husband
     name    ; name of the person
     husband ; husband's name
)

(literalize Wife
     name    ; name of the person
     wife    ; wife's name
)

(p Initialize
   {(Start) <initialize>}
   -->
   (make Father ^name mohan ^father rama)
   (make Mother ^name lata ^mother alka)
   (make Wife ^name hari ^wife lata)
   (make Wife ^name rama ^wife alka)
   (make Father ^name uma ^father hari)
   (remove <initialize>))
```

Figure 2.3a: The Family Example in OPS

```
(p p1
   (Father ^name <y> ^father <x>)
   (Wife ^name <x> ^wife <z>)
   -(Mother ^name <y> ^mother  <z>)
  -->
   (make Mother ^name <y> ^mother  <z>))

(p p2
   (Mother ^name <y> ^mother <x>)
   (Husband ^name <x> ^husband <z>)
   -(Father ^name <y> ^father  <z>)
  -->
   (make Father ^name <y> ^father  <z>))

(p p3
   (Wife ^name <y> ^wife <x>)
   -(Husband ^name <x> ^husband <y>)
  -->
   (make Husband ^name <x> ^husband <y>))

(p p4
   (Husband ^name <y> ^husband <x>)
   -(Wife ^name <x> ^wife <y>)
  -->
   (make Wife ^name <x> ^wife <y>))

(p p5
   (Father ^name <z> ^father <x>)
   (Mother ^name <z> ^mother <y>)
   -(Husband ^name <y> ^husband <x>)
  -->
   (make Husband ^name <y> ^husband <x>))

(p p6
   (Father ^name <z> ^father <x>)
   (Mother ^name <z> ^mother <y>)
   -(Wife ^name <x> ^wife <y>)
  -->
   (make Wife ^name <x> ^wife <y>))
```

Figure 2.3b: The Family Example in OPS (Contd.)

2.8 Rule Matching

As can be seen from the description given above, there is a large amount of *matching* which has to be done in each cycle of the inference engine. To determine the conflict set, the system has to match every antecedent of every rule against each working memory element. If there are n rules in the rule base with m antecedents each on an average and there are k elements in the working memory, the number of matches required is $m * n * k$.

The system also needs to ensure that the same variables in different antecedents of a rule are bound to the same values, i.e., the variable bindings are consistent.

Therefore it is not surprising that the bulk of the time taken by most rule based systems is in this matching process and in consistency checking. Some systems have been observed to spend more than nine-tenths of their total execution time on these.

Since rule matching is such a crucial phase of the inference procedure of rule based systems, a number of efficient algorithms have been worked out to speed up this phase. The most commonly used one is the RETE algorithm which was developed by CL Forgy and is used in a number of commercial systems like OPS5. A brief description of RETE is given in the next section.

2.8.1 RETE Matching Algorithm *

As mentioned earlier the RETE matching algorithm developed by Forgy is used in the OPS family of programming languages. It is also used in a number of expert system tools like Automated Reasoning Tool (ART) from Inference Corporation and Knowledge Craft from the Carnegie Group. The RETE algorithm computes the conflict set in a rule based system. In this section we give a brief description of the two main features of the RETE algorithm. There are other features which also contribute to the efficiency of the algorithm but are not as significant, and so are left out. There have also been modifications to the RETE algorithm in systems where it has been used. For more information on the RETE algorithm readers can refer to [Forgy, 1982].

Avoiding Exhaustive Search of Working Memory

In expert systems, the working memory contents change very little from cycle to cycle (for large systems the change is likely to be less than 1%). Therefore, if the set of rules which match the working memory are computed in each cycle, the system does a lot of unnecessary matching. This is especially true when the size of the working memory is very large (which is the case for most practical applications).

In order to avoid this, the RETE algorithm stores match information from the previous cycle and uses it to compute the set of rules which match in the current cycle. The set of rules can be determined by using the match information in the previous cycle along with the changes to the working

* This section may be omitted during a first reading.

memory which occurred in the previous cycle. For example, if a working memory element is removed in the previous cycle, some of the rules which matched that element need to be removed from the conflict set. Similarly if a new working memory element is added, additional rules would need to be added to the conflict set. The effect of a modifying an element is similar to the effect of a deletion followed by an addition.

Avoiding Exhaustive Search of the Rule Base

Another common thing in expert systems, is that a number of rules in the rule base often share common antecedents. Therefore when rule matching takes place, a working memory element has to be often matched with the same antecedent in different rules. The RETE algorithm tries to reduce this inefficiency by indexing all rules which have the same antecedent.

The way this is done is by processing the rule base beforehand. A rule compiler processes all the rules and creates a network. Antecedents are represented as nodes in the network. If two rules have the same antecedent, the two rules will share a node in the network. Given a working memory element the RETE algorithm can use the network to determine all rules in which the element will match an antecedent.

In the example we discussed in Section 2.4, suppose the system added a new element husband(hari, lata) to the working memory in the previous cycle. The system could determine that the only rules which will match this element in the current cycle are p2 and p4 and avoid checking the other rules. Therefore only one third of the rules in the example have to be checked.

In essence the RETE algorithm creates a network representation, takes the changes to working memory, uses the network and maps these to changes in the conflict set.

2.9 Meta-Rules

Rules meant to control the inference mechanism of a rule based system are called meta-rules. These rules can be either domain-specific or domain-independent. These rules are called *meta*-rules, because they, decide what domain rules should be applied next or in what order they should be applied.

For instance, instead of treating all rules as being at the same level, it may be useful to partition the rules. Meta-rules can be used to select a partition of rules depending on the contents of the working memory.

An example of a meta-rule from the MYCIN expert system [Davis et al, 1977] is:

> **If** the site of the culture is one of the nonsterile sites
> **and** there are rules which mention in their premise a previous
> organism which may be the same as the current organism
> **then** it is definite that each of them is not going to be useful

Meta-rules thus help in increasing the efficiency of systems and in some cases make the knowledge more understandable.

2.10 Advantages of Rule Based Systems

Some of the advantages of rule based systems are:

- Homogeneity

  Because of the uniform syntax, the meaning and interpretation of each rule can be easily analyzed.

- Simplicity

  Since the syntax is simple, it is easy to understand the meaning of rules. Domain experts can often understand the rules without an explicit translation. Rules therefore can be self-documenting to a good extent.

- Independence

  While adding new knowledge one need not be worried about where in the rule base the rule is added, or what the interactions with other rules are. In theory, each rule is an independent piece of knowledge about the domain. However, in practice, this is not completely true, as we shall see in the next section.

- Modularity

  The independence of rules leads to modularity in the rule base. You can create a prototype system fairly quickly by creating a few rules. This can be improved by modifying the rules based on performance and adding new rules.

- Knowledge is Separated from Use and Control

  The separation of the rule base from the inference engine separates the knowledge from how it is used to solve the problem. This means that the same inference engine can be used with different rule bases and a rule base can be used with different inference engines. This is a big advantage over conventional programs where data and control are intermixed.

- Procedural Interpretations

  Apart from declarative interpretation, rule based systems have procedural interpretations also, which enable them to be viewed as computational models.

2.11 Drawbacks of Rule Based Systems

In spite of the advantages mentioned above, rule based systems have their own drawbacks. Some of the drawbacks are listed below:

- Lack of Methodology

  There is no methodology (i.e., systematic procedure), yet for creating rule based systems. Most systems are built based on intuition, prior experience, and trial and error.

- Interaction among Rules

  An advantage of the rule based representation was stated to be the relative independence of the different pieces of knowledge. However, in many systems you cannot assume that the rules do not interact among themselves. In certain cases, ignoring rule interaction could lead to unexpected results.

- Opacity

  Rule based systems provide no mechanism to group together related pieces of knowledge. This makes any structure/relationships in the domain opaque in the rule base. Other representations which overcome this problem are described in Chapter 4.

- Lack of Structure

  The simplicity of rules leads to the drawback that all rules are at the same level. In many domains it would be useful to have rules at different levels in a hierarchy, but the pure production system model does not support this.

- Representing Procedural Tasks

  Some tasks which can be easily represented in terms of procedural representations are not very easy to represent using rule based representations. For instance, it requires about 10 rules to model two column subtraction using production rules. This sort of modelling is useful because it gives you a fine grain analysis of human problem solving. However if you just wanted a system to solve subtraction problems, using production rules is tedious, to say the least.

- Inefficiency

  As mentioned earlier a large amount of time is taken in each cycle to match applicable rules in the rule base. For large rule bases, this often leads to inefficiencies. However, there is work going on in creating more efficient matching algorithms. In addition, structuring the rule base can also lead to increase in efficiency. The matching procedure can also be made faster by doing the matching in parallel, if a parallel machine is used.

2.12 Good Domains for Rule Based Systems

Rule based systems have been used for a variety of applications such as medical diagnosis and machine fault troubleshooting, etc. It would be difficult to list all such domains. However here are some characteristics of domains which can meaningfully use the rule based framework [Davis and King, 1984].

- Where the Knowledge is Diffuse

  For example, clinical medicine is a good domain because it consists of a large number of facts which are more or less independent of each other. In contrast, mathematics has a strong theoretical base and a set of inter-related principles which need to be applied to solve problems.

- Where Processes are Representable as Independent Actions

  If a process consists of a set of independent actions, there need not be much communication among the rules and therefore such processes are ideal for the rule based framework. For example, a domain like medical diagnosis is a good domain, as opposed to a domain like accounting.

- Where Knowledge can be Easily Separated from its Use

  The periodic table in chemistry provides knowledge about the different elements. This knowledge is independent of how it is used. In contrast, a recipe for a food item not only says what the ingredients required are but also how they should be mixed. Just knowing the ingredients would not be of much use!

2.13 Limitations of Rule Based Systems

Some of the limitations of present day rule based systems are [Alty, 1987]:

- Inadequate Explanations

  Most expert systems provide a facility by which they can provide an explanation of their *behaviour* to the user. The explanation facilities provided by rule based systems are rather limited. Most systems give you the sequence of rules which have been fired by the system. There are two basic problems here. Firstly, human experts do not give explanations by describing rules they have applied. They give an explanation using examples and analogies which will help the user understand things better. Secondly, the knowledge encoded in the rules is knowledge which a user conversant with the domain can understand. If a novice uses the system, he may not be able to understand the same explanation.

  Therefore the explanation facilities offered by rule based systems are only adequate if the user and the expert are at the same level of knowledge. A more elaborate description of explanation facilities provided in rule based systems is given in Chapter 6.

- Implicit Control Knowledge

  As mentioned earlier the order of the rules is immaterial in an ideal rule based system. However in real applications, this is not always true. For example, in the MYCIN expert system, the rules are ordered so that particular rules are selected from the conflict set. Therefore the reason why a rule is being considered is *implicit* and not *explicit*. Therefore, the system will not be able to explain why a particular rule is being fired.

- Inadequacies of Rules as a Knowledge Representation Formalism

  There are a number of domains where experts use more than just rules to come to decisions. For example, in electric circuit fault diagnosis the expert has a model of the electric circuit which helps the expert diagnose the fault in the system. Such information cannot be easily represented in terms of rules.

2.14 Conclusion

In this chapter we have looked at various facets of rule based systems. In practical systems the rule base is often augmented with means of representing uncertainty. Ways of representing uncertainty will be described in Chapter 5.

Most commercial expert systems use the rule based formalism to represent knowledge. Examples of some systems will be presented in Chapter 7. The widespread use of this representation indicates that, despite some disadvantages, rule based representations are still the most popular type of representations used.

Summary

- Rules are one of the major types of knowledge representation formalisms used in expert systems.
- The main components of a typical rule based system are: the working memory, the rule base and the inference engine.
- The working memory contains information about the particular instance of the problem being solved.
- The rule base is a set of rules which represent the problem solving knowledge about the domain.
- A rule contains a set of conditions (antecedents) and a set of conclusions (consequents).
- The inference engine uses the rule base and the working memory to derive new information.
- An instantiation of a rule is a set of working memory elements which will satisfy the antecedent(s) of that rule.
- The set of rules which can be fired is called the conflict set. One of these rules has to be selected for firing.
- The conflict resolution strategy decides which rule from the conflict set should be selected in each cycle of the system.
- Three common conflict resolution strategies are: specificity, recency and refractoriness.
- The two inferencing strategies used in expert systems are forward chaining and backward chaining. Forward chaining is essentially data driven. Backward chaining on the other hand is a goal driven strategy.
- OPS is a popular class of rule based languages used for forward chaining applications.
- Rule matching is one of the most expensive parts of the inferencing process in a rule based system. The RETE algorithm is an efficient algorithm for rule matching.

- Rule based systems have declarative interpretations compared to conventional programs which have only procedural interpretations.
- The two major inferencing strategies are: backward chaining and forward chaining.
- Meta-rules can be used to improve efficiency and understandability of rule based systems.
- While current rule based systems have limitations, the rule based representation is still the most popular knowledge representation mechanism used in expert systems.

Exercises

1. Give examples of knowledge which would be required in an expert system context, but cannot be adequately represented using rules.

2. A forward chaining production system consists of the following production rule:

   p1: **If** hungry(X) **and** serves(Y, Z) **then** goes-to(X,Y)

   which represents the knowledge that **if** X is hungry **and** (restaurant) Y serves (food) Z **then** X goes to Y.

   The working memory (WM) contains the following elements with a tag indicating the chronological order in which they were put into the WM.

WM element	Time tag
serves(balaji, dosa)	4
serves(taj, puri)	6
serves(balaji, idli)	3
hungry(gopal)	1
hungry(sekar)	5
hungry(sunita)	2

   If we ignore the time tags and just use the textual order of the working memory elements, the first set of working memory elements which match the antecedents of rule p1 are:

Antecedent	WM element
hungry(X)	hungry(gopal) (by binding X to gopal)
serves(Y,Z)	serves(balaji, dosa) (by binding Y to balaji and Z to dosa)

   Therefore, the system can derive the following new information:

   goes-to(gopal, balaji).

   List out all instantiations based on the WM given above. If you are using recency in your conflict resolution strategy, which instantiation would qualify to fire? What will be added to the WM as a result?

3. Suppose we add the following rule to the rule base in problem 2:

   p2: **If** hungry(X) **and** likes(X,Z) **and** serves(Y, Z) **then** goes-to(X,Y)

   which represents the knowledge that:

   If X is hungry
   and X likes (food) Z
   and (restaurant) Y serves Z
   then X goes to Y.

   The WM contains the following facts in addition to those given in the previous problem.

   likes(sekar, puri)
   likes(sunita, idli)
   likes(sunita, dosa)

   Given these WM contents and the two rules and using specificity as the conflict resolution strategy what are the resulting candidate instantiations? Use textual order of facts in the WM for conflict resolution within this resulting set. Which instantiation gets selected and what fact is added to the WM as a result?

4. Use the two rules and the WM contents given in the problems 2 and 3 above. Assume specificity and refractoriness are used in that order for conflict resolution. Assuming that the system cycles until there are no more rules to fire, what will the working memory contain finally? In particular, which restaurant will Gopal go to?

   If we use refractoriness first, and specificity next, for conflict resolution, will Gopal go to a different restaurant?

5. As mentioned in this chapter, rule matching is an expensive part of the inferencing process. Can you think of ways in which this process could be made more efficient? See if your solution can reduce the amount of rule matching in the family example given in Section 2.4.

6. Suppose we add the following two rules to the rule base given in Section 2.4.

   p7: **If** husband(X,Y) **then** male(X)
   p8: **If** wife(X,Y) **then** female(X)

   What would be the additional facts the system would derive? What order would they be derived in?

7. For each of the following tasks, determine whether the inference engine should use forward chaining or backward chaining. Justify your answers.

   (a) designing the layout of a house
   (b) diagnosing a patient for a disease
   (c) determining the classification of a certain chemical substance
   (d) troubleshooting a piece of electronic equipment

8. Give two or three meta-rules in any domain of your choice.

3. Clausal Form Logic

3.1 Introduction

In an expert system, knowledge about the domain under consideration has to be encoded in a form which can be used by the computer. The computer has to make use of the encoded knowledge to solve problems in the domain. A number of standard formalisms (or languages) have been developed to encode the knowledge about a domain. Some of the well understood formalisms are: *if-then rules, clausal form logic, semantic networks* and *frames*. This chapter deals with clausal form logic. Chapter 2 has dealt with if-then rules. Chapter 4 will deal with semantic networks and frames.

If-then rules are the most popular formalism used in expert systems. The popularity of if-then rules can be explained by their conceptual simplicity, availability of efficient implementations and reasonably high expressive power. The expressive power indicates the ability to represent different types of knowledge.

Mathematical logic can be used for the purpose of knowledge representation in AI systems. In fact, the idea of using logic as the basis for AI systems is almost as old as AI itself. The advent of the logic based programming language PROLOG ([Bratko, 1986] and [Clocksin and Mellish, 1989]) and its adoption by the Japanese as the fundamental implementation language for their fifth generation computer systems project have given a fillip to the research on logic based AI ([Jackson et al, 1989] and [Genesereth and Nilsson, 1987]).

The expressive power of logic based knowledge representation languages is much better than that of if-then rules. But, at present, logic based knowledge representation languages are not as popular among expert system implementors as if-then rules. This is due to lack of efficient implementations. We believe that in the near future the efficiency of implementations of logic based languages will improve and they will be used extensively in implementing expert systems and other AI systems.

We can identify three aspects of a logic – *syntax, semantics* and *proof procedures*. The syntax defines the permissible structures. It specifies the basic structures and rules to combine them to form complex structures. The semantics defines the meanings of the structures. The proof procedures are the formal inference mechanisms which can be used to infer new (logically valid) structures out of the existing structures. The newly inferred structures should be *truth preserving* – that is, their meanings should be consistent with the meanings of the existing structures.

In this chapter, we deal with Clausal Form Logic (CFL) [Richards, 1989] as an example for logic based knowledge representation languages. We look at CFL from a knowledge representation point of view. We define its syntax in terms of symbols related to traditional computer science. From a knowledge representation point of view, resolution refutation is the prime inference mechanism. In this chapter, we restrict our discussion to the SLD-refutation procedure, which is based on the principle of resolution refutation.

We keep the presentation simple, concise and introductory in nature. Readers who are interested in knowing more about the computational aspects of logic can refer to [Chang and Lee, 1973] or [Lloyd, 1984].

The outline of this chapter is as follows: Section 3.2 describes the syntax of CFL, Section 3.3 introduces the semantics, Section 3.4 describes a proof procedure, and Section 3.5 lists the significant pieces of research in computational logic.

3.2 Syntax of CFL

This section explains the syntax of CFL, first considering the basic structures and then the formation of complex structures out of the basic structures. The *syntax* of a knowledge representation language specifies a set of basic structures, and rules to combine them to form complex structures. Typically, in formal languages, the complex structures are called *sentences*. We use computer-related symbols (that is, upper-case and lower-case alphabets, digits and special characters) as the basic ingredients to define CFL structures.

First we will study the basic structures which are allowed in CFL.

3.2.1 Basic Structures

The four types of basic structures in CFL are: *constant, variable, functor* and *literal*. The complex structure, which can be formed out of these basic structures, is called a *clause*. The generic name for a constant, variable, or functor is *term*. [1]

Constant

A constant begins with a lower-case letter or a digit, followed by zero or more upper-case or lower-case letters, underscores or digits.

A few examples of constants are: bombay, india, ravi and 25. The following are *not* constants: AirPort, NEXTSTATE and _TEMP.

Variable

A variable begins with an upper-case letter, followed by zero or more upper-case or lower-case letters, underscores or digits.

[1] We do not cover existential terms so as to keep the discussion as simple as possible.

A few examples of variables are: Person, Port and X1. The following are *not* variables: area, base and _TEMP.

Functor

A functor begins with a functor-name followed by an opening bracket, followed by one or more arguments (separated by commas), followed by a closing bracket. A functor with n arguments is called an n-ary functor. The general form of a functor is:

functor-name(argument$_1$, argument$_2$, ..., argument$_n$)

The form of the functor-name is the same as that of a constant. Each of the arguments of a functor can be either a constant, variable, or functor. Because a functor can be an argument of another (possibly same) functor, one can construct nested (or recursive) functors.

A few examples of functors are: date(11, 12, 1992), date(11, december, 1992) and name(kowalski, robert). The following are *not* functors: Address(NCST, Gulmohar_Road, Juhu, Bombay) and Circle(origin(10, 10), radius(5)).

Literal

A literal is the basic element of a clause in CFL. A clause is composed of zero or more literals.

A literal begins with a predicate-name followed by an opening bracket, followed by one or more arguments (arguments separated by commas), followed by a closing bracket. A literal with n number of arguments is called an n-ary literal. The general form of a literal is:

predicate-name(argument$_1$, argument$_2$, ..., argument$_n$)

The form of a predicate-name is the same as that of a constant. The arguments of the literal have to be constants, variables or functors.

Though, syntactically, a functor and a literal look alike, semantically, they are totally different entities – a functor is used to represent an object, whereas a literal is used to represent a relationship among objects.

A few examples of literals are: parent(ravi, gopal), part_of(bombay, india) and greater_than(25, 17). The following are *not* literals: Capital(delhi, india) and Head(ComputerScience, Vivek).

Now let us see how the basic structures are combined to form clauses.

3.2.2 Clause

A clause is formed by combining a number of literals using *connectives*. The permissible connectives are $\Leftarrow$ (*implication*), $\wedge$ (*and*), and $\vee$ (*or*).

A clause begins with a consequent-part, followed by an implication and then an antecedent-part. The general form of a clause is:

consequent-part $\Leftarrow$ antecedent-part

This is to be read as: *the antecedent-part implies the consequent-part* or as *the consequent-part is true if the antecedent-part is true.*

The consequent-part consists of a number of literals connected together by or's. The general form of the consequent-part is:

$\text{literal}_1 \vee \text{literal}_2 \vee \ldots\ \text{literal}_N$

The antecedent-part consists of a number of literals connected together by and's. The general form of the antecedent-part is:

$\text{literal}_1 \wedge \text{literal}_2 \wedge \ldots\ \text{literal}_N$

Example

A few examples for clauses are:

mother(X, Y) $\vee$ father(X, Y) $\Leftarrow$ parent(X, Y)
mother(X, Y) $\Leftarrow$ parent(X, Y) $\wedge$ female(X)

The following are *not* clauses:

capital(City, Country) $\wedge$ part_of(City, Country)
$\Leftarrow$ contains(Country, City)
PartOf(Person, cabinet) $\Leftarrow$ PartOf(Person, government) $\square$

Positive, Negative and Empty Clauses

There are three special types of clauses: positive clause, negative clause and empty clause.

A positive clause is a clause with no antecedent-part. The general form of a positive clause is:

consequent-part $\Leftarrow$

Examples of positive clauses are:

daughter_of(indira, nehru) $\Leftarrow$
son_of(rajiv, indira) $\Leftarrow$

A negative clause is a clause with no consequent-part. The general form of a negative clause is:

$\Leftarrow$ antecedent-part

Examples of negative clauses are:

$\Leftarrow$ capital_of(bombay, india)
$\Leftarrow$ male(Person) $\wedge$ female(Person)

An empty clause is a clause in which both the consequent-part and antecedent-part are absent. So, what is left? The implication! An empty clause is represented by $\Leftarrow$.

Now that we have seen what are the valid structures of CFL, let us see how these structures can be used to represent something meaningful. The semantics (or meaning) aspects of CFL are described in the next section.

3.3 Semantics of CFL

The semantics of a knowledge representation language deal with the meanings of the structures. In this section, we first give the semantics of CFL in an intuitive manner. We later provide an introductory treatment of the formal aspects of the semantics.

3.3.1 Intuitive Semantics

Our primary interest is to use CFL as a knowledge representation language. CFL is a language which can be used to describe a world of our choice. We want to use the clauses of CFL to encode (or represent) the facts (or knowledge) of a certain domain (or world).

A domain can be viewed as consisting of individuals (or objects) with relationships among them. An object can be a human being, animal, inanimate object, or some abstract object. The nature of the relationships among the objects can be simple or complex depending on the domain.

A relationship can be between two objects. For example, consider the following sentence: *The daughter of Jawaharlal Nehru is Indira Gandhi.* In this sentence, the symbols *Jawaharlal Nehru* and *Indira Gandhi* represent two individuals. And, *daughter of* represents the relationship of being a daughter.

A relationship can be for a single object. For example, consider the following sentence: *Indira Gandhi is the Prime Minister.* In this sentence, the symbol *Indira Gandhi* is used to represent the individual. And, *is the Prime Minister* is used to represent the relation (property) of being the Prime Minister.

More complex relationships can exist among three or more objects.

We will take an example domain and illustrate how CFL can be used to represent the knowledge in the domain. The domain that we are interested in is the famous Nehru dynasty.

A part of the knowledge available about the Nehru dynasty can be stated as follows: *Jawaharlal Nehru was the son of Motilal Nehru. He was also the father of Indira Gandhi. Indira's husband was Feroz Gandhi. Indira had two sons: Rajiv Gandhi and Sanjay Gandhi. Rajiv and Sanjay were married to Sonia and Maneka, respectively. Sonia has a daughter, Priyanka, and a son, Rahul. Maneka has a son Varun.*

How can CFL be used to represent what has been stated above? It can be done in the following way *(in general, any given knowledge can be represented using CFL in more than one way).*

$$\mathsf{son\_of(jawahar, motilal)} \Leftarrow \qquad (\mathsf{C1})$$
$$\mathsf{father\_of(jawahar, indira)} \Leftarrow \qquad (\mathsf{C2})$$
$$\mathsf{husband\_of(feroz, indira)} \Leftarrow \qquad (\mathsf{C3})$$
$$\mathsf{son\_of(rajiv, indira)} \Leftarrow \qquad (\mathsf{C4})$$
$$\mathsf{son\_of(sanjay, indira)} \Leftarrow \qquad (\mathsf{C5})$$
$$\mathsf{husband\_of(rajiv, sonia)} \Leftarrow \qquad (\mathsf{C6})$$
$$\mathsf{husband\_of(sanjay, maneka)} \Leftarrow \qquad (\mathsf{C7})$$
$$\mathsf{daughter\_of(priyanka, sonia)} \Leftarrow \qquad (\mathsf{C8})$$
$$\mathsf{son\_of(rahul, sonia)} \Leftarrow \qquad (\mathsf{C9})$$
$$\mathsf{son\_of(varun, maneka)} \Leftarrow \qquad (\mathsf{C10})$$

The above clauses are self-explanatory. Their informal meanings can be analysed by considering a couple of them. The clause C1 represents the fact that Jawaharlal Nehru is a son of Motilal Nehru. The constants, jawahar and motilal stand for Jawaharlal Nehru and Motilal Nehru, respectively. The predicate son_of represents the relation of somebody being a son of somebody else. Similarly, the clause C2 represents the fact that Jawaharlal Nehru is the father of Indira Gandhi. In clause C2, the constants jawahar and indira stand for the individuals Jawaharlal Nehru and Indira Gandhi, respectively. The two occurrences of the constant jawahar, in C1 and C2, stand for the same individual, Jawaharlal Nehru.

Some general knowledge, relevant to any family, can be represented as follows:

$$\text{wife\_of}(X, Y) \Leftarrow \text{husband\_of}(Y, X) \quad \text{(C11)}$$
$$\text{son\_of}(X, Y) \Leftarrow \text{son\_of}(X, Z) \wedge \text{husband\_of}(Y, Z) \quad \text{(C12)}$$
$$\text{son\_of}(X, Y) \Leftarrow \text{son\_of}(X, Z) \wedge \text{wife\_of}(Y, Z) \quad \text{(C13)}$$
$$\text{father\_of}(X, Y) \vee \text{mother\_of}(X, Y) \Leftarrow \text{son\_of}(Y, X) \quad \text{(C14)}$$
$$\text{father\_of}(X, Y) \vee \text{mother\_of}(X, Y) \Leftarrow \text{daughter\_of}(Y, X) \quad \text{(C15)}$$

The clause C11 represents the fact that if a person is the husband of someone, then that someone is the wife of the person. The same knowledge can also be stated as follows: if Y is the husband of X, then X is the wife of Y. In this clause, the variables, X and Y, can stand for any individual (not *one particular* individual) in the domain.

Now let us see how the meaning of the structures can be defined in a mathematical way.

3.3.2 Formal Semantics

In this section, we concentrate on defining, in a mathematical sense, the meanings for the structures. This section assumes on the part of reader a certain amount of mathematical inclination and in particular, some basic understanding of set theory.

The ability to define the meanings of structures mathematically is an important property of CFL. This property enables the use of CFL to encode the facts and knowledge of a domain precisely. In fact, this is one of the primary reasons behind the attempt to use CFL as a knowledge representation language for AI systems.

The mathematical definition of meanings for the structures has another important consequence. It enables one to determine (by means of *some* procedure) if a newly inferred structure has a *logically* valid meaning or not. The newly inferred structure is logically valid if it is *truth preserving* – that is, the meaning of the structure is consistent with the meanings of the existing structures.

In defining formal semantics, we emphasise abstract domains (or worlds), consisting of abstract objects and abstract relationships. The abstract world might correspond to the physical world, but we do not necessarily have to limit ourselves to *reality*.

The formal semantics of CFL can be defined in terms of an interpretation and a model. The primary purpose of the definition is to find out a world in which all the clauses are *true*. However, for some set of clauses, it may *not* be possible to define such a world.

An interpretation consists of two parts – a domain of discourse, and an interpretation function. The domain of discourse is the set of objects which we are considering. For example, the domain of discourse might be the set of all staff of an organisation or the set of all colours. The interpretation function is a function which maps each of the basic structures into an object, relation, or function in the domain of discourse.

The rest of this section consists of definitions of some concepts and examples using these concepts. We define concepts such as interpretation, term assignment, truth value of a clause, model, logically valid consequence, etc.

Definition

Let S be a set of clauses. Let $I = \langle W, \mathrm{f} \rangle$ be an interpretation for S. W is called the *domain of discourse* and f is called the *interpretation function*. Then,

for any constant, a_i, occurring in S, $\mathrm{f}(\mathsf{a}_i)$ is an object in W;

for any n-ary functor-name, $\mathsf{f}_i{}^n$, occurring in S,
$\mathrm{f}(\mathsf{f}_i{}^n)$ is a mapping from all n-tuples in W onto another object in W;

for any n-ary predicate-name, $\mathsf{p}_i{}^n$, occurring in S,
$\mathrm{f}(\mathsf{p}_i{}^n)$ is an n-ary relation in W. □

Definition

Let S be a set of clauses, and $I = \langle W, \mathrm{f} \rangle$, be an interpretation for S. Let T be a term assignment for S in I. Then,

for any constant, a_i, occurring in S, $T(\mathsf{a}_i) = \mathrm{f}(\mathsf{a}_i)$;

for any variable, X_i, $T(\mathsf{X}_i)$ is an object in W;

for any n-ary functor, $\mathsf{f}_i(\mathsf{t}_1, \mathsf{t}_2, ..., \mathsf{t}_n)$, in S,
$T(\mathsf{f}_i(\mathsf{t}_1, \mathsf{t}_2, ..., \mathsf{t}_n)) = \mathrm{f}(\mathsf{f}_i)(T(\mathsf{t}_1), T(\mathsf{t}_2), ..., T(\mathsf{t}_n))$. □

Definition

Let S be a set of clauses, and $I = \langle W, \mathrm{f} \rangle$, be an interpretation for S. Let T be a *term assignment* for S with respect to I. The meaning (that is, the *truth value*) of a clause C in S, with respect to I and T can be stated as follows:

if C is of the form $\mathsf{p}(\mathsf{t}_1, \mathsf{t}_2, ..., \mathsf{t}_n)$, then C is true
if $\langle T(\mathsf{t}_1), T(\mathsf{t}_2), ..., T(\mathsf{t}_n) \rangle$ is a member of $\mathrm{f}(\mathsf{p})$;
otherwise, C is false.

if C is of the form $C_1 \wedge C_2 \wedge ... \ C_m$, then C is true
if all C_i'*s* are true;
otherwise, C is false.

if C is of the form $C_1 \vee C_2 \vee ... \ C_m$, then C is true
if at least one of C_i'*s* is true;
otherwise, C is false.

if C is of the form $C_c \Leftarrow C_a$, then C is *false*
if C_a is true and C_c is false;
otherwise, C is true.

if C is of the form $C_c \Leftarrow$, then C is true if C_c is true;
otherwise, C is false.

if C is of the form $\Leftarrow C_a$, then C is true if C_a is false;
otherwise, C is false.

if C is of the form $\Leftarrow$, then C is false. □

Definition

Let S be a set of clauses and $I = \langle W, \mathrm{f} \rangle$ be an interpretation for S. Let

C be a clause in S. Then, C is true with respect to I if for all possible term assignments with respect to I, the clause is true. □

Example

Consider the following clauses:

$$\mathsf{p(num1,\ num2)} \Leftarrow \qquad \text{(C1)}$$
$$\mathsf{p(X,\ Z)} \Leftarrow \mathsf{p(X,\ Y)} \wedge \mathsf{p(Y,\ Z)} \qquad \text{(C2)}$$

Let $I = <W$, f$>$ be an interpretation. Let W be a finite set of natural numbers, for example let $W = \{3, 4, 5, 7, 8\}$.

Let us define the interpretation function as follows:

f(num1) $= 3$
f(num2) $= 4$
f(p) = the *less than* relation

Under this interpretation, I, and the term assignment, X = 3, Y = 5, Z = 7, we can easily verify that both C1 and C2 are true. More importantly, we can also verify that both C1 and C2 are true with respect to I for all possible term assignments. *Readers are urged to convince themselves that this is so.* Thus, we conclude that both C1 and C2 are true with respect to I. □

Example

Consider the following clauses:

$$\mathsf{p(person1,\ person2)} \Leftarrow \qquad \text{(C1)}$$
$$\mathsf{p(X,\ Y)} \Leftarrow \mathsf{p(Y,\ X)} \qquad \text{(C2)}$$

Consider an interpretation, $I = <W$, f$>$, in which the domain is a set of four people, $W = \{$Ravi, Rakesh, Vivek, Gopal$\}$. Assume that the relation of friendship is defined by the following set $\{$<Ravi, Rakesh>, <Rakesh, Gopal>, <Rakesh, Ravi>$\}$.

Let us define f as follows:

f(person1) = Rakesh
f(person2) = Gopal
f(p) = the *friendship* relation

We can easily verify that both C1 and C2 are true with respect to I and the term assignment, X = Ravi, Y = Rakesh. But, we can also verify that with respect to I and the term assignment, X = Gopal, Y = Rakesh, only C1 is true and C2 is false. Thus, we conclude that the set of clauses, consisting of C1 and C2, is not true with respect to I. □

Definition

Let S be a set of clauses. If an interpretation can be defined in which all the clauses of S are true, then S is said to be consistent. If no interpretation can be defined in which all the clauses of S are true, then S is said to be inconsistent. A consistent set of clauses is called satisfiable. An inconsistent set of clauses is called unsatisfiable. □

Definition

Let S be a set of clauses. If an interpretation can be defined in which all the clauses are true, then the interpretation is called a model of S. □

Definition

If a clause can be proved to be true in all possible interpretations, then it is called a valid clause. □

Definition

Let S be a set of clauses, and C be any clause. The clause C is a logically valid consequence of S, iff C is true in every interpretation I which is a model for S. □

Example

Consider the following two clauses:

$$\mathsf{p(num1,\ num2)} \Leftarrow \qquad \text{(C1)}$$
$$\mathsf{p(num2,\ num3)} \Leftarrow \qquad \text{(C2)}$$

Consider an interpretation, $I = <W, \text{f}>$, in which the domain is a finite set of natural numbers, say, $W = \{2, 4, 5, 3, 7\}$. Assume that the standard relationships hold among the numbers. Let f be an interpretation function defined as follows:

$\text{f}(\mathsf{num1}) = 7$
$\text{f}(\mathsf{num2}) = 5$
$\text{f}(\mathsf{num3}) = 3$
$\text{f}(\mathsf{p}) =$ the *greater than* relation

We can verify that C1 and C2 are true in I. That is, I is a model for C1 and C2. Now, we want to verify if the following clause logically follows from C1 and C2:

$$\Leftarrow \mathsf{p(num1,\ num3)} \qquad \text{(C3)}$$

We can easily verify that under the interpretation, I, C3 is not true. *Readers are urged to convince themselves that this is so.* So, I is not a model for the set of clauses consisting of C1, C2 and C3. Thus, C3 does not follow logically from C1 and C2. □

When CFL is used to encode the knowledge of a domain, problem solving in the domain reduces to inferring logically valid conclusions using certain inference mechanisms. In general, an inference mechanism is a collection of inference rules, which can be applied to infer new clauses (or conclusions). This can be performed either by human beings or computers. For computers to infer new conclusions, the inference mechanism has to be as simple and deterministic as possible – ideally, it should be one rule, which the computer can blindly apply to infer new clauses. The inference rule should also be efficient, so that computers do this fast; otherwise, it is not of much use. In the following section, we will describe a proof procedure called SLD-refutation procedure. We outline the resolution refutation principle, followed by a discussion on how to resolve two clauses and how to obtain the negation of a clause. We then give the actual SLD-refutation procedure with an example to concretise the ideas.

3.4 Proof Procedure for CFL

Proof procedures deal with inferring new structures (or expressions) from the set of given expressions. A proof procedure is a *formal mechanism* – that is, its manipulation of structures is based entirely on the forms of the structures, and *not* on the meanings of structures. In CFL, it is a mechanism to infer new clauses from a set of given clauses. To be of any use, the proof procedure must be such that the newly inferred clause must be a logical consequence of the original set of clauses.

Proof procedures are a collection of rules, which taking the forms of the given expressions into consideration, allow one to infer logically valid expressions. From a knowledge representation point of view, the inference rule of interest is called resolution [Robinson, 1965]. The AI community is interested in resolution refutation rule, which is based on goal driven application of resolution. In this chapter, we study one type of resolution refutation called SLD-refutation procedure.

In the following section, we study the principle behind resolution refutation, how to resolve two clauses and how to obtain the negation of a clause. Then using these ideas, we study how SLD-refutation works. We will first look at the principle behind resolution refutation.

3.4.1 The Resolution Refutation Principle

Intuitively, the principle behind resolution refutation is as follows: to prove a statement to be true, prove the negation of the statement to be not true. By doing so, the original statement will be proved to be true. But why should the negation of a statement be proved to be not true rather proving that the statement itself is true? This is due to reasons of computational efficiency. In computational logic, it is usually more efficient to prove the falsity of a statement than the truth of it.

Let S be a set of clauses which is consistent (that is, it has a model). Assume that we want to verify if a clause, G, logically follows from S. To do that, we determine the negation of G. Let the negation be $\neg G$. Add $\neg G$ to the set S to get the set $S' = S \cup \{\neg G\}$. If S' is inconsistent, then we can conclude that G logically follows from S. This is because, if S' is inconsistent, then it means that for any interpretation in which S is true, $\neg G$ is false; that is, G is true. Thus, we conclude that G logically follows from S if S' is inconsistent.

On the other hand, if S' is consistent, then G is not a logical consequence of S. Because, if S' is consistent, then it means that for *some* interpretation in which S is consistent, $\neg G$ is also true; that is, G is false. Which implies that if S' *is* consistent, then G is not a logical consequence of S.

We will now see how to resolve two clauses.

3.4.2 Resolving Two Clauses

Resolution is a procedure which takes two clauses as input and outputs a new clause (called the *resolvent*). Resolution consists of the following stages: standardising-apart the variables (if necessary) in the input clauses, finding the most general unifier of unifiable literals, and resolving the clauses through the unifiable literals. We will explain each of these three stages with the help of an example.

Consider the following two clauses:

$$\Leftarrow \mathsf{p(X,\ a)} \tag{C1}$$

$$\mathsf{p(X,\ Z)} \Leftarrow \mathsf{q(X,\ Z)} \wedge \mathsf{r(Y,\ Z)} \qquad \text{(C2)}$$

First let us see what we mean by standardising-apart two clauses.

Standardising-apart

If the input clauses to the resolution procedure contain occurrences of common variables, then they need to be standardised apart. To do this, we locate the common variables in the two clauses, C1 and C2. We then replace the common variables occurring in *one* of the clauses consistently by other variables which do not occur in either of the clauses.

In our example, X is the common variable which occurs in both C1 and C2. We choose to replace all occurrences of X in C1 by a new variable, say, U. This results in the new clause C3:

$$\Leftarrow \mathsf{p(U,\ a)} \qquad \text{(C3)}$$

Note that C3 preserves the meaning of C1.

Now let us see when two literals are unifiable, and how to get the most general unifier of two unifiable literals.

Finding the Most General Unifier

At the heart of resolution-based inference lies the unification procedure. Unification is so crucial from the implementation point of view, that an efficient implementation of it is of considerable research interest.

Unification is a procedure which takes two literals, p1 and p2, as input and attempts to make them identical by binding the variables in either literal appropriately. If successful, it outputs a *unifier*, θ, which is a set of *substitutions*. Otherwise, it fails.

If θ is a unifier of p1 and p2, then the application of θ to p1 and p2, makes them identical. We express this as follows: $\mathsf{p1}[\theta] = \mathsf{p2}[\theta]$. In general, there are many possible unifiers for a given pair of literals. A unifier θ is called the most general unifier if no subset θ' (except θ) of θ is a unifier of p1 and p2. That is, $\mathsf{p1}[\theta'] \neq \mathsf{p2}[\theta']$ for all θ' such that $\theta' \subset \theta$, and $\theta' \neq \theta$.

In our example, p(U, a) and p(X, Z) are unifiable. These two literals can be made identical by substituting U for X and a for Z. Thus the unifier is the set of substitutions $\{\mathsf{U/X},\ \mathsf{a/Z}\}$. We can also verify that this unifier is the most general unifier. *Readers are urged to verify that this is so.*

The reason behind standardising apart the clauses and determining the most general unifier (rather than just a unifier) is to keep the resultant resolvent as general as possible so that it is as *productive* as possible for further resolutions.

Now we will see how to determine if two clauses are resolvable and resolve them.

Resolve

After the variables of the two clauses are standardised apart and the most general unifiers of two unifiable literals are determined, the clauses are resolved to get the resolvent. The resolvent is obtained by the following procedure: apply the most general unifier to both the input clauses; remove the unifiable literals from the input clauses; merge the (left over) consequent-parts of the input clauses to get the consequent-part of the resolvent; merge the (left over) antecedent-parts of the input clauses to get the antecedent-part of the resolvent.

In our example, the clauses C2 and C3 are resolved to get the following resolvent,

$$\Leftarrow \mathsf{q(U,\ a)} \wedge \mathsf{r(Y,\ a)} \qquad \text{(C4)}$$

with the most general unifier $\{\mathsf{U/X},\ \mathsf{a/Z}\}$.

Now we present some ideas on how to obtain negations of CFL clauses.

3.4.3 Negation of a Clause

As we noted above, from a knowledge representation point of view, we would be interested in verifying if a clause (or *assertion*) logically follows from a set of clauses (or *knowledge-base*). In order to do that, the negation of the assertion has to be determined. The negated assertion is called the *goal*. In this section, we do not give any formal techniques to determine the negations of assertions. Instead, we restrict ourselves to simple assertions and explain the negation process through examples. The following examples give an idea of how to determine the negation of simple assertions.

Consider the following assertions:

$$\Leftarrow \mathsf{mortal(socrates)} \qquad \text{(C1)}$$
$$\mathsf{partof(bombay,\ india)} \Leftarrow \qquad \text{(C2)}$$

The negations C3 and C4 respectively of C1 and C2, are given below:

$$\mathsf{mortal(socrates)} \Leftarrow \qquad \text{(C3)}$$
$$\Leftarrow \mathsf{partof(bombay,\ india)} \qquad \text{(C4)}$$

Consider the following assertion:

$$\Leftarrow \mathsf{mortal(socrates)} \wedge \mathsf{eternal(socrates)} \qquad \text{(C5)}$$

The negation of C5 is *not* one clause, but two clauses. They are the following:

$$\mathsf{mortal(socrates)} \Leftarrow \qquad \text{(C6)}$$
$$\mathsf{eternal(socrates)} \Leftarrow \qquad \text{(C7)}$$

Intuitively, clause C5 states that it is not true that Socrates is a mortal and also an eternal being. The negation of this statement is that Socrates is a mortal and also an eternal being. And, this is what is stated by clauses C6 and C7 together.

Now consider another assertion:

$$\mathsf{successful(ravi)} \Leftarrow \mathsf{hardworking(ravi)} \qquad \text{(C8)}$$

The negation of C8 consists of two clauses C9 and C10:

$$\Leftarrow \mathsf{successful(ravi)} \qquad \text{(C9)}$$
$$\mathsf{hardworking(ravi)} \Leftarrow \qquad \text{(C10)}$$

The clause C8 says that if Ravi is hardworking, then Ravi is successful. The negation of this statement is that Ravi is hardworking and Ravi is not successful. This is what is stated by the clauses C9 and C10.

Now that we have covered concepts like the principle behind resolution refutation, how to resolve two clauses and how to determine the negation of clauses, let us see how the SLD-refutation procedure actually works.

3.4.4 SLD-refutation Procedure

There are a number of proof procedures based on the principle of resolution refutation. One of the most popular inference mechanisms is called SLD-refutation. The name SLD-refutation is an abbreviated form of Linear resolution with Selection function for Definite clauses [Lloyd, 1984]. The SLD-refutation is based on the concepts described in Sections 3.4.1, 3.4.2 and 3.4.3. A brief description of the mechanism is given below:

Let S be a set of clauses which has a model. Assume that, we want to verify if a clause G_i logically follows from S. Let $\neg G_i$ be the negation of G_i. Let $\neg G_i$ be of the form $\Leftarrow \mathsf{A}_1 \wedge \mathsf{A}_2 \wedge ... \wedge \mathsf{A}_k \wedge ... \wedge \mathsf{A}_n$. $\neg G_i$ is called the goal, and $\mathsf{A}_i s$ are called subgoals. Let C_i be a clause in S, and be of the form $\mathsf{A} \Leftarrow \mathsf{B}_1 \wedge \mathsf{B}_2 \wedge ... \wedge \mathsf{B}_m$.

Assume that A_k and A are unifiable, and let θ_i be the most general unifier. Then, the resolvent of $\neg G_i$ and C_i, say, G_{i+1}, is of the form, $\Leftarrow \mathsf{A}_1 \wedge \mathsf{A}_2 \wedge ... \wedge \mathsf{B}_1 \wedge \mathsf{B}_2 \wedge ... \wedge \mathsf{B}_m \wedge ... \wedge \mathsf{A}_n$. The resolvent is the new goal for the SLD-refutation procedure.

A subgoal of G_{i+1} is selected and resolved with a clause from S. It may be worth noting that, this subgoal of G_{i+1} should only be resolved with a clause from S, and *not* from any of the earlier resolvents obtained so far. This process is continued till an empty clause is inferred. If an empty clause is inferred, SLD-refutation succeeds, that is, the set $S \cup \{\neg G_i\}$ is proved to be inconsistent. Thus, G_i is shown to be a logical consequence of S.

At some point of time, if no subgoal of the current goal is resolvable with any of the clauses in S, then a mechanism called *backtracking* is used to try an alternative possible resolution for an already resolved subgoal. *The* backtracking *mechanism is illustrated in the example below.* If all possibilities are exhausted for all subgoals, and still an empty clause is not inferred, then SLD-refutation fails. This means that, SLD-refutation could *not* come to the conclusion that G_i is a logical consequence of S.

Let us take a simple example to concretise the ideas presented above about the SLD-refutation procedure. This example also illustrates how the mechanism of backtracking is actually used to try alternative resolutions.

Example

Consider the following two clauses:

$$\mathsf{p(X, Y)} \Leftarrow \mathsf{q(Y, X)} \quad \text{(C1)}$$
$$\mathsf{p(X, X)} \Leftarrow \quad \text{(C2)}$$

Assume that we want to verify if

$$\mathsf{p(a, a)} \Leftarrow \quad \text{(C3)}$$

logically follows from C1 and C2. By negating the conclusion, C3, we get the goal:

$$\Leftarrow \mathsf{p(a, a)} \quad \text{(C4)}$$

The subgoal p(a, a) of C4 is unifiable with the literal p(X, Y) of C1 and the literal p(X, X) of C2. We choose the first option now. Thus, by resolving C4 and C1, we get the new goal as

$$\Leftarrow \mathsf{q(a,\ a)} \qquad \text{(C5)}$$

with the most general unifier {a/X, a/Y}.

The clause C5 is not resolvable with any of the original clauses, that is, neither C1 nor C2. At this point, *backtracking* is used to detect the alternate option available for resolving the subgoal p(a, a) of C4. As we noted earlier, the alternative option is to unify the literal p(X, X) of clause C2 with the subgoal p(a, a) of C4. Thus, the clauses C4 and C2 are resolved. The resolvent this time is

$$\Leftarrow \qquad \text{(C6)}$$

By inferring an empty clause, C6, the SLD-refutation has been able to establish that C3 follows logically from C1 and C2. □

The SLD-refutation procedure as you can see is goal driven. That is, the next step in the procedure is determined by the current goal. This property is very important from a computational point of view – goal driven inference procedures are computationally efficient. The inference procedure employed in the logic based programming language PROLOG is based on SLD-refutation procedure.

In the next section, we list some of the important research works in computational logic. These works form the foundations for logic based knowledge representation languages.

3.5 Major Research Works

The work on logic based AI is based on computational logic. Some of the major pieces of work in the field of computational logic have been the following. Robinson in [Robinson, 1965] came up with a novel proof procedure called resolution procedure for automatic theorem proving in predicate logic. Kowalski in [Kowalski, 1974] and [Kowalski, 1979a] showed how predicate logic can be used as a programming language and as a knowledge representation language. Also, he demonstrated in [Kowalski, 1979b] how logic together with a control mechanism (inference procedure) could be used to implement algorithms. Warren in [Warren, 1983] came up with efficient implementation techniques for PROLOG.

The programming language PROLOG can be compared with CFL as follows. A PROLOG program is a set of Horn clauses. Horn clause logic allows only one literal to be present in the consequent-part of a clause; thus, PROLOG also allows only one. PROLOG, because of efficiency reasons, allows only one negative clause to be present in the knowledge-base. This clause is the query (or goal). Since PROLOG allows only the query to be a negative clause, explicit negation of facts is not possible in it. It makes a closed-world assumption, and realises negation using negation-by-failure [Lloyd, 1984]. The negation-by-failure principle states that a goal is false if it cannot be proved to be true.

The inference procedure followed in PROLOG is based on the SLD-refutation procedure. For efficiency reasons PROLOG provides a pruning mechanism which allows the user to prune parts of the proof tree which are not going to lead to a solution.

The Japanese adopted PROLOG as the kernel-level language for the fifth generation computer systems project. Over the years PROLOG has gained popularity among the software implementors. A large number of applications have been developed using it [Sterling, 1990].

Summary

- Logic enables to precisely encode the knowledge of a domain.
- Logic based knowledge representation languages have not been very popular among the implementors of expert systems because of lack of efficient implementations.
- CFL is a simple and powerful logic based knowledge representation language.
- In CFL, the user creates a set of clauses to encode the knowledge about a domain.
- An inference procedure is used to solve problems in the domain using the encoded knowledge.
- SLD-refutation is based on the principle of resolution refutation and is used in PROLOG.
- Pioneering works in the field of computational logic includes:
  - Robinson's resolution principle,
  - Kowalski's work showing how logic can be used as a computer problem solving tool, and
  - Warren's work on the efficient implementation of PROLOG.
- The popular AI programming language PROLOG is closely related to CFL.

Exercises

1. Encode the knowledge expressed in the following passage using CFL.

   India is called a subcontinent. It has a number of neighbours. On the north-west, it has Pakistan. Along the north-east border, it has Nepal, Bhutan and China. On the east, it has Burma and Bangladesh. On the south is Sri Lanka. Maldives lies to the west of India. Sri Lanka and Maldives are island countries.

2. Express in English, the knowledge encoded in the following clauses. Assume the intended meanings.

   $\mathsf{kr\_language(rule)} \Leftarrow$ (C1)
   $\mathsf{kr\_language(cfl)} \Leftarrow$ (C2)
   $\mathsf{efficient(rule)} \Leftarrow$ (C3)
   $\mathsf{expressive(rule)} \Leftarrow$ (C4)
   $\Leftarrow \mathsf{efficient(cfl)}$ (C5)
   $\mathsf{expressive(cfl)} \Leftarrow$ (C6)
   $\mathsf{popular(X)} \Leftarrow \mathsf{kr\_language(X)} \wedge \mathsf{efficient(X)} \wedge \mathsf{expressive(X)}$ (C7)

3. Consider the following clauses,

   $\Leftarrow \mathsf{p(X, Y)} \wedge \mathsf{p(Y, X)}$ (C1)
   $\mathsf{p(X, Z)} \Leftarrow \mathsf{p(X, Y)} \wedge \mathsf{p(Y, Z)}$ (C2)

   Come up with a model for these clauses.

4. Consider the previous problem, and the model you have come up with. Try to find out if the following clause logically follows from the clauses in the previous problem.

   $\mathsf{p(X, Y)} \Leftarrow \mathsf{p(Y, X)}$ (C3)

5. Consider the following clauses:

   $\mathsf{p(X, Z)} \Leftarrow \mathsf{q(X, Y)} \wedge \mathsf{p(Y, Z)}$ (C1)
   $\mathsf{p(X, X)} \Leftarrow$ (C2)
   $\mathsf{q(a, b)} \Leftarrow$ (C3)

   Use SLD-refutation to determine if the following clause logically follows from them.

   $\mathsf{p(a, b)} \Leftarrow$ (C4)

6. Attempt to unify the following pairs of expressions. If they are unifiable determine their most general unifiers. If they are not, explain why they will not unify.

   (a) p(X,Y) and p(a,Z)
   (b) p(X,X) and p(a,b)
   (c) ancestor(X,Y) and ancestor(ram, father (ram))
   (d) ancestor(X, father (X)) and ancestor(ram, mohan)
   (e) q(X) and q(X,a)
   (f) p(X,a,Y) and p(Z,Z,b)

7. Express the following knowledge in CFL.

   Anything that flies is a bird unless it has teeth.

8. Consider the following set of axioms:

   $\mathsf{dog(X)} \Leftarrow \mathsf{beagle(X)}$
   $\mathsf{pet(X)} \Leftarrow \mathsf{dog(X)}$
   $\mathsf{cat(X)} \Leftarrow \mathsf{siamese(X)}$

$\mathsf{pet(X)} \Leftarrow \mathsf{cat(X)}$
$\Leftarrow \mathsf{pet(pluto)}$
$\Leftarrow \mathsf{pet(tom)}$
$\mathsf{beagle(pluto)} \Leftarrow$
$\mathsf{siamese(tom)} \Leftarrow$

Is this set of axioms consistent? Explain your answer.

4. Structured Representations: Semantic Networks and Frames

4.1 Why Structured Representations

In the last few chapters, we have looked at logic and rule based representations of knowledge. While these formalisms have a number of advantages, they also have their share of disadvantages and limitations (see Sections 2.11 and 2.13).

The major argument against these essentially syntactic formalisms is that they lack structure. They are essentially *flat* representations, where all information is put together without any significant organization. Even if there is a logical structure inherent in the domain information, there are no mechanisms in these formalisms to use or reflect this structure in the representations.

There are also other arguments against the use of these formalisms. There is an attempt at minimalism in logic, with a resulting economy in the use of logical operators and variables. Often, this can be at the cost of the richness of the representation. Ambiguous interpretations may also result out of this.

Therefore, we need other mechanisms which allow one to structure knowledge. Further, we require that these mechanisms be unambiguous and rich in expressive power.

In this chapter, we study early *triple* mechanisms (a triple is a 3-tuple), semantic networks and frame systems. The concepts discussed are illustrated with examples. The attempt is to show the reader some knowledge representation formalisms that are available, other than logic and rules, and discuss how such representations can be compared.

As we go from triples to semantic networks to frames, we will notice a shift from purely syntactic mechanisms (for representing information) towards more semantic mechanisms. There is more meaning to the manner in which knowledge is represented, though the representation scheme itself becomes slightly complicated to manipulate and use.

4.2 Object-Attribute-Value Triples

The simplest way to represent any information is by some sort of association. In English, you say, for example:

The patient's name is Lata and her age is 42.
Her complaint is chest-pain.

This can be very easily represented as a set of triples as follows:

patient	name	lata
patient	age	42
patient	complaint	chest-pain

The first item in each line is the name of the object. The second item is the name of an attribute of the object, and the third is the value of the attribute. Thus these are *object-attribute-value* triples.

Practically any information can be reduced to such triples. For example, if we want to say

The patient complained of chest-pain, tiredness and blackouts.

we can use the following triples to represent this information:

patient	complaint	chest-pain
patient	complaint	tiredness
patient	complaint	blackouts

Alternatively, we can represent this information using a list of values:

patient complaint (chest-pain tiredness blackouts)

It may seem that only binary relations (relations of the form R(A,B), connecting a pair of items) may be represented in triples. However, any n-ary relation R(A1, A2, ...An) can be broken up into a set of n related binary relations, using a fixed event or object specifier. Thus, if R(A1 ... An) describes an event E, we can have the relations

R1(A1, E)
R2(A2, E)
...
Rn(An, E)

which can be represented easily as triples of the form:

E Rk Ak

Thus the common item E binds all the triples together.

For example, to represent

Sunita gave a book to Uma

we can have a relation give, with three arguments:

give(Who, Whom, What)

so that, if we identify the book that is given as book3, we have

give(Sunita, Uma, book3)

To represent this as triples, we need to identify the giving event as some event, say G1, and break the *give* relation into the following relations:

give-giver(Sunita, G1).
give-receiver(Uma, G1).
give-object(book3, G1).

Early AI programmers realised the need for such mechanisms. The programming language SAIL (Stanford Artificial Intelligence Language), which incorporated the language LEAP [Feldman and Rovner, 1969], provided for object-attribute-value triples. The programming language LISP and its derivatives have provided property lists, which are essentially triples in another form: *object – property – value.*

In SAIL, triples could be indexed so that any partial specification of a triple could be used to retrieve all matching triples. Thus

patient complaint $*$

would match the triples:

patient complaint chest-pain
patient complaint blackouts

LISP has a limited version of this; given the object and the attribute, the value may be obtained. Thus it is easy to access information contained in triples.

However, the problem with these structures is that we have to repeat the name of the object many times. Also, because of its very general nature, there is no clear domain structure that is apparent in a bunch of such triples. In this respect, triples are similar to logic or rule based formalisms. When a large body of knowledge is to be represented, you would like it to be more structured, and easily modifiable. In addition, you would like to have well-defined methods of using this stored knowledge.

Semantic Networks and Frames are two attempts to solve this problem. They are discussed in detail in this chapter.

4.3 Semantic Networks

Semantic Networks were created essentially out of the seminal work of Ross Quillian [Quillian, 1967;1968] which dealt with a simulation program intended to be able to "compare and contrast the meanings of arbitrary pairs of English words".

The basic idea behind Semantic Networks (or Semantic Nets) is simple: the meaning of a concept is defined by the manner in which it is related (or linked) to other concepts.

A semantic network consists of nodes representing concepts interconnected by different kinds of associative links.[1] These links represent binary relations between a pair of concepts. N-ary relations have to be broken up into binary relations, as discussed in the previous section. Using standard graph notation, these networks can be displayed and manipulated.

A semantic network differs from a graph in that there is meaning (semantics) associated with each link. This semantics comes usually from the fact that the links are labelled with standard English words.

Quillian distinguished between *type* nodes and *token* nodes in a semantic network. Type nodes defined classes of objects, while tokens denoted specific instances of a type of object. Thus, *human* or *woman* would be examples of types, while Gita would be a token, or a particular instance, of the type *woman.* **a-kind-of, is-a, a-part-of**, and **has-part** were the main relations

[1]For this reason, semantic networks have also been called *associative networks.*

linking type and token nodes. These are discussed in the section on inheritance (Section 4.3.2) that follows.

Thus a semantic network consists of a set of related concepts, linked to form a graph. These concepts can be very general information about the world (constituting *world knowledge*), or can be concepts specific to some particular domain.

Concepts which are semantically closer will be closer to each other in the network, while concepts which have no real connection will be far apart. The path(s) through the network connecting two concepts are quite significant; they indicate how the two concepts are related, and how closely they are related. To understand how to locate the connecting path(s), we need to understand the notion of search in semantic networks.

4.3.1 Search in Semantic Networks

Quillian's aim was to compare and contrast two given word concepts, say W1 and W2. The user can traverse the semantic network from the nodes W1 and W2, trying to detect an intersection between the two concepts. The traversal is done by spreading activation along the network. Following normal graph traversal procedure, to avoid wasteful moves, it is useful to mark each node visited in the network.

Thus, we start with concept W1, and take one step forward in a direction which takes us closer to W2. (This is an important step, which could be directed, say, by domain-dependent heuristics. It is very easy to get lost in any network.) This starts the path from the W1 side. To contain the movement, we then switch to the W2 side, and take a step towards W1, starting the W2 path. As the search proceeds, each node that is visited is tagged with its starting node name (the name of its *patriach*), and concepts are expanded from the W1 side and the W2 side alternately.

If a node is visited from the W1 side, then it is given a W1 tag; else it is given a W2 tag. If the node is already tagged, it is either because we have reached an intersection, or because there is a loop in the traversal. Thus, when you are expanding the W1 path, if you visit a node which already has a W1 tag, you are looping. If it is a W2 tag, an intersection of the two paths has been found!

The path connecting the two concepts defines the commonality between the two concepts. Each concept in the path is related in some way to both W1 and W2.

Figure 4.1 shows the relation between concepts cry and comfort. The path from cry goes through cry2[2], which is the sense of cry meaning *to make a sad sound*, and through the concept sad. The path from comfort goes through the concept comfort3, relating to *making less sad*, and then through the concept sad. Thus, there is an intersection at the concept sad. Quillian's program could generate natural language sentences to talk about the relation between these concepts. For this example, the following sentences were generated:

[2]Numerals are used to distinguish between different senses of the same word.

```
Comparing:  CRY, COMFORT
Intersection node:  SAD
(1) CRY IS AMONG OTHER THINGS TO MAKE A SAD SOUND.
(2) TO COMFORT3 CAN BE TO MAKE2 SOMETHING LESS2 SAD.
```

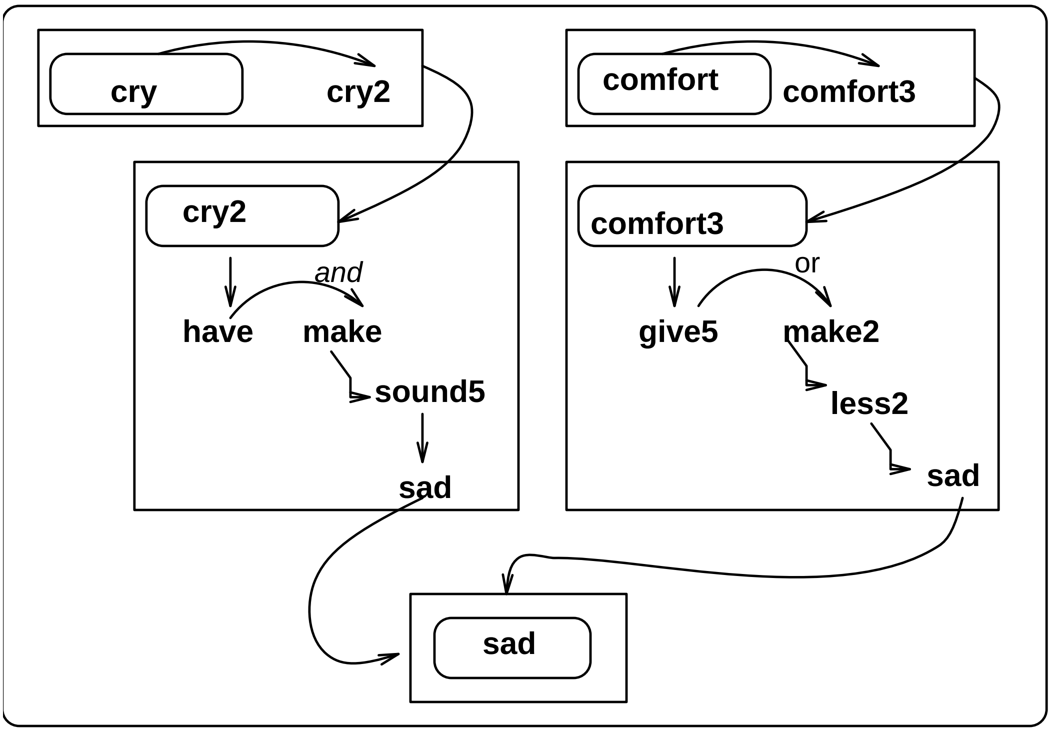

Figure 4.1: Relation between concepts cry and comfort. The intersection node is the concept sad.

This use of *intersection search* for inference with the help of a spreading activation mechanism is one of the innovative features of Quillian's semantic network formalism.

4.3.2 Inheritance

The notion of *inheritance* was defined in an extension to Quillian's early work. We will discuss this fundamental idea in this section.

For any domain, we can visualise a type hierarchy. Types inherit properties from their *supertypes*. Such hierarchies are normally represented using links called **a-kind-of** and **is-a** (or **instance-of**). The link **a-kind-of** (often abbreviated to *ako*) relates types (of objects/events) to sub-types (of objects/events). **is-a** relates types to instances.[3] Thus we can say:

[3] In AI literature, there are differences in the way these relations are named. At some level, it may not be important what we call these relations. But for consistency, in this chapter, we will use only the meanings defined here for these relationships.

Professor **a-kind-of** *Person*
Anaka **is-a** *Professor*

One basic idea that is true of such a hierarchy is that the properties of an object at a particular level include *all* the properties defined for it at that node, *plus all those defined for all the supertypes of that node*. Thus Anaka will have all the properties of *Professor*; in addition, Anaka will also *inherit* properties of the *Person* node (as well as those of the ancestors of that node). Similarly, if Alka **is-a** *Girl* and *Girl* is **a-kind-of** *Human* and *Human* is **a-kind-of** *Mammal*, Alka will have the properties defined at the level of the Alka node. In addition, all the properties of the nodes *Girl*, *Human* and *Mammal* will be inherited by Alka.

Inheritance has several advantages. Information that is common to all objects of a class needs to be represented just once. Avoiding duplication has other beneficial side-effects. It becomes easier to modify information, without the need to explicitly propagate the changes to other levels. It also reduces or eliminates the problem of maintaining a consistent set of facts. Thus inheritance leads to a compact representation.

So far, we have discussed objects with at most one supertype. In general, you could have more than one parent (supertype) for a given class (or object of a class). In this case, the child class or object may inherit information from all its supertypes, leading to *multiple inheritance*. There are several issues raised by multiple inheritance – what to do if there is conflicting information, for instance.

There have been reaction-time studies done [Collins and Quillian, 1972] to see if inheritance is indeed a mechanism used by humans. For example, consider a human who is told:

Birds have wings.
A crow is a bird.
A crow is black.

Given that the more general properties are stored higher in the generalization hierarchy, does it take more time to realise that crows have wings, than that crows are black? If so, perhaps a mechanism such as inheritance is being used. There is conflicting evidence about this, but the results seem to indicate the plausibility of humans using inheritance mechanisms.

These reaction time experiments also gave rise to the notion of *semantic distance* – the number of nodes, for example, between any two concepts (or the number of links/arcs to be traversed between any two concepts). This acts as a measure of the "closeness" of the concepts. Two concepts which are closely linked, for example, cry and sad, will have a very small semantic distance.

4.3.3 How Semantic Networks Have Been Used

In this section, we will take a brief look at some early systems that used semantic networks, and the types of links that were used in these networks. See [Findler, 1979] for additional examples.

In his program SCHOLAR [Carbonell, 1970], Jaime Carbonell (Sr.) used a semantic network to define South American geography. SCHOLAR distinguished between *concept* units (e.g. Latitude) and *example* units (e.g. Argentina), just as *types* are distinguished from *tokens*. Properties were named after attributes. SCHOLAR also had (special) named links such as **superc**, **superp** and **supera** (standing for superconcept, superpart and superattribute respectively).

Winston used semantic networks to represent structural descriptions in a program called ARCH [Winston, 1975]. Winston's program could infer concepts such as an arch, given examples of an arch and several near-misses (things that look like an arch, but are not.) The program constructs structural descriptions of a concept and refines the description, based on new examples. These descriptions are essentially represented as semantic networks. The program used primitives such as **left-of**, **right-of**, and **supported-by**. Relations could be defined in terms of other relations. For example:

> **left-of** *is opposite* **right-of**

Another topic which has attracted research attention is the problem of scoping in semantic nets. In logic, it is easy, using the existential (there-exists) and universal (for-all) quantifiers, to make statements such as:

> All dogs have tails.
> There exists at least one white elephant.
> Every student failed in at least one test.

One problem associated with quantification is the notion of *scoping*. Briefly, this deals with the problem of defining the scope of a quantified variable. If a statement S is true for all values of a variable V, we say that S is the *scope* of V. In a logical formula, this may be represented using terms in parentheses. ForAll X (F) indicates that the scope of the quantifier ForAll is the expression F. If we wish to allow quantification of statements in semantic networks, how can we do it?

Hendrix used *partitioned* nets [Hendrix, 1975] to represent quantified information, such as "Every dog has bit a postman". In this example, the scope of a statement is constrained by the partition(s) in which it is applicable.

Semantic networks have been used to classify knowledge (using hierarchies etc.). They have also been used to represent "world knowledge", that is, knowledge that is universal and common across several domains. In addition, they have been used for natural language understanding and generation [Simmons and Bruce, 1971].

Rumelhart and his colleagues used nodes to represent *concepts, events* and *episodes* (sequences of events) [Rumelhart et al, 1972]. They used the **is-a** relation to represent both type-token and subset relations. Many other links were not motivated or explained. Inheritance concepts were not always clear. These were typical of problems other semantic network implementations suffered from.

The utility of the graph notation lies in the fact that certain kinds of inference directly map on to graph-searching techniques. For example, locating attributes or values of attributes reduces to traversal through the graph, across links such as **is-a**. In addition, the path between concepts shows how they are related.

However, the ad-hocism about relations and their interpretation has been criticised, and identified as one of the major weaknesses of semantic networks.

There have been other areas of concern in semantic networks, which we briefly survey next.

4.3.4 Criticisms of Semantic Networks

There have been several criticisms of semantic networks, prominent among them a paper entitled *What's in a Link* [Woods, 1975].

Till Woods' paper, the semantics of networks were very unclear. Woods argued that it is important to look at the nature of primitives proposed. By pointing out problems and inadequacies of semantic networks, he stimulated more formal work in this area.

Consider the following example: Rocks can be classified as igneous rocks, metamorphic rocks and basaltic rocks. They can also be classified by colour: black rocks, white rocks, orangish-red rocks etc. This can be represented in the semantic network formalism by a set of nodes connected to their super-type node rock, as shown in Figure 4.2.

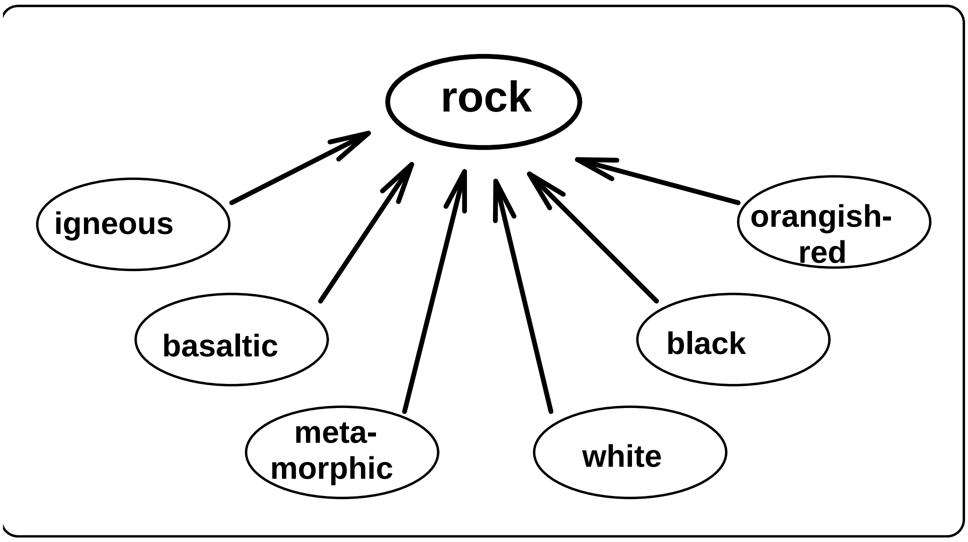

Figure 4.2: A Hierarchy of Rock-types

Does this mean that there are as many types of rocks as there are labels attached? This would be a rather simplistic assumption, besides being wrong! However, many semantic networks in use assumed that the number of child-links from an object node represented the number of types you could classify the object into.

The type dimension (basaltic, igneous etc.) and the colour dimension are not really independent of each other. Is it not logical that the dimensions of representation be independent? And should not dimension information be reflected in the semantic net formalism? Since little thought was paid to the dimensions of representation, in most semantic networks, these questions went unanswered.

For these and other reasons, semantic networks, in the form described above, were seen to be limited in utility. Frames (described in the next section) were seen as one logical step towards a more semantic representation of information.

However, there has been continuing interest in semantic networks, with attempts to go beyond the limitations of early semantic network systems. Brachman surveyed the growth and use of semantic networks and provided one of the first formal specifications of the role of semantic networks [Brachman, 1979]. [Sowa, 1991] is a recent collection of excellent articles on the semantic network formalism and its applications.

4.4 Frames

Frames are general purpose representations of a cluster of facts or properties, which describe a prototypical situation or object in detail. In a sense, they are equivalent to semantic nets, where each node has considerable structure.

The general idea behind frames is very old. However, the first major discussion of the utility of frames in Artificial Intelligence can be attributed to Marvin Minsky's seminal paper [Minsky, 1975;1981], which had a profound impact on knowledge representation and AI. The discussion of frames in this chapter will be based primarily on Minsky's paper.

We start with an intuitive motivation for frames.

4.4.1 Why Frames are Useful

Consider a traffic accident situation. Suppose we hear that a car has collided with a truck, leaving two people dead. There are various questions that immediately arise, for example:

- *Where was this accident?*
- *When?*
- *Was anyone injured?*
- *How did the accident occur?*

Other related question could also be asked:

- *Whose fault was it?*
- *Could it have been prevented?*

When we hear of an accident, we immediately take a lot of things for granted, such as the fact that usually accidents cause damage to people and property. We also know that accidents are due to negligence, or sometimes weather conditions, or some strange combination of causes. We need some mechanism to store the standard (default) knowledge that people have about events and objects.

Traffic accidents are one particular type of accidents. Aircraft accidents are another type. Whatever is true of accidents in general, will be true of traffic accidents in particular. In addition, there may be more specific issues that arise. Similarly, whatever is true of an accident, and more than that too, will be true of aircraft accidents. Thus we say that traffic accidents and aircraft accidents (among other types of accidents) *inherit* all the properties of the prototypical accident.

We need a formalism to represent information such as that about accidents. This formalism must also provide mechanisms such as inheritance. Frames satisfy these requirements, and provide other facilities as well.

In the next section, we examine the properties of frames. We will use several examples to illustrate properties of frames. In particular, we will use a set of frames (see Figures 4.3a and 4.3b) on geography to illustrate the concepts being discussed. The notation used is an informal variant of a popular frame language – the exact syntax of the language is not critical.

Define a concept world*, which is* **ako** thing*.*

Define	world ako thing
type	:value concept

Define a concept continent*. Parameters* area*,* perimeter *and* number-of-political-units *of this concept are integer-valued, with at most one value.*

Define	continent apo world
type	:value concept
area	:type integer :min 1 :max 1
perimeter	:type integer :min 1 :max 1
number-of-political-units	:type integer :min 1 :max 1

Define an instance Asia *of* continent*. Parameters* area*,* perimeter *and* number-of-political-units *of this concept have specific integer-values.*

Define	Asia instance-of continent
area	:value 2000000
perimeter	:value 1000000
number-of-political-units	:value 20

Figure 4.3a: A Geography Frame System

4.4.2 Properties of Frames

Frames can be used to describe objects, classes of objects (for example, a room) or typical events such as a birthday party, or a traffic accident.

As we shall see, frames can be used, among other things, to

- store such prototypical information,

Country *is defined as a-part-of (***apo***)* continent*. Certain attributes of* continent*, such as* area, perimeter *etc., are "masked" off. They are not visible or used at this or lower levels. Note that this masking is done using another attribute of the frame.*

Define	country apo continent
type	:value concept
do-not-inherit	:value (number-of-political-units area perimeter)
capital-city	:type string
language	:type string
president	:type string
prime-minister	:type string
major-religions	:type string :min 1 :max 5
government-type	:type string :min 1 :max 1
major-rivers	:type string :min 1 :max 9
neighbours	:type list :min 1 :max 6

Define an instance of country*, with valid input for all parameters.*

Define	India instance-of country
type	:value instance
capital-city	:value "New Delhi"
language	:value "English Hindi"
president	:value "Shankar Dayal Sharma"
prime-minister	:value "PV Narasimha Rao"
population	:value 880000000
major-religions	:value "Hinduism" "Islam" "Christianity" "Buddhism" "Jainism"
government-type	:value "Secular Socialist Democratic Republic"
major-rivers	:value "Ganga" "Jamuna" "Godavari" "Sutlej" "Krishna" "Narmada" "Cauvery" "Brahmaputra"
neighbours	:value ((west Pakistan Arabian-Sea) (east Bangladesh Burma Bay-of-Bengal) (north China Nepal Himalayas) (south Indian-Ocean))

Figure 4.3b: A Geography Frame System (contd.)

- represent default information,
- provide multiple perspectives (viewpoints) of a situation, and
- provide for analogies.

The power of the frame formalism derives from all these features. Each of these features is discussed in this section.

Subsumption and disjointness are properties of frame-like structured types. These properties are not available in formalisms such as semantic nets, where all types are atomic. One class is said to subsume (encompass) another, if every element of the subsumed class is a member of the subsuming class. For example, the class *Human* subsumes the class *Woman*. If two classes are disjoint, no element of one class can be an element of the other class. For example, the classes *Woman* and *Man* are disjoint. Subsumption and disjointness relations help in having efficient and meaningful procedures to manipulate and use frames.

Structure of Frames

A frame can be thought of as a network of nodes and relations, just like a semantic network. One part of the frame has fixed information, consisting of information that is always true of the situation represented. The other part of the frame has many *slots*, which are slots that must be filled with specific values, instances or data. A frame can be looked upon as a form to be filled. A generic event or object is described by such a form, and an instance of the event or object is described by a (partially) filled-in form. Thus, to represent all available information about an object or an event, one would have to create an instance of (i.e., instantiate) an appropriate class frame.

To represent world knowledge, or knowledge about a domain, we need several frames related in an intricate fashion. A *frame system* provides for this.

In the geography frame system shown in Figures 4.3a and 4.3b we have defined, in turn, the world, continents, countries and states. Asia is defined as a specific instance of a continent, and India as an instance of a country. This simple top-level view of the hierarchy is shown in Figure 4.4.

Nested Frames

Slots may be filled with values that are themselves frames. Thus frames can be nested; for example, a *house* frame can have embedded *room* frames in it. Nested frames let you view the world at different levels of granularity. At some level, a *house* is just a set of *rooms*; at another level, each room has certain characteristics, such as size, height, colour of the walls, number and type of windows and doors etc.

Inheritance and Masking in Frames

Frames inherit properties and values of properties just as nodes in semantic networks inherit properties. That is, all the properties and values that are

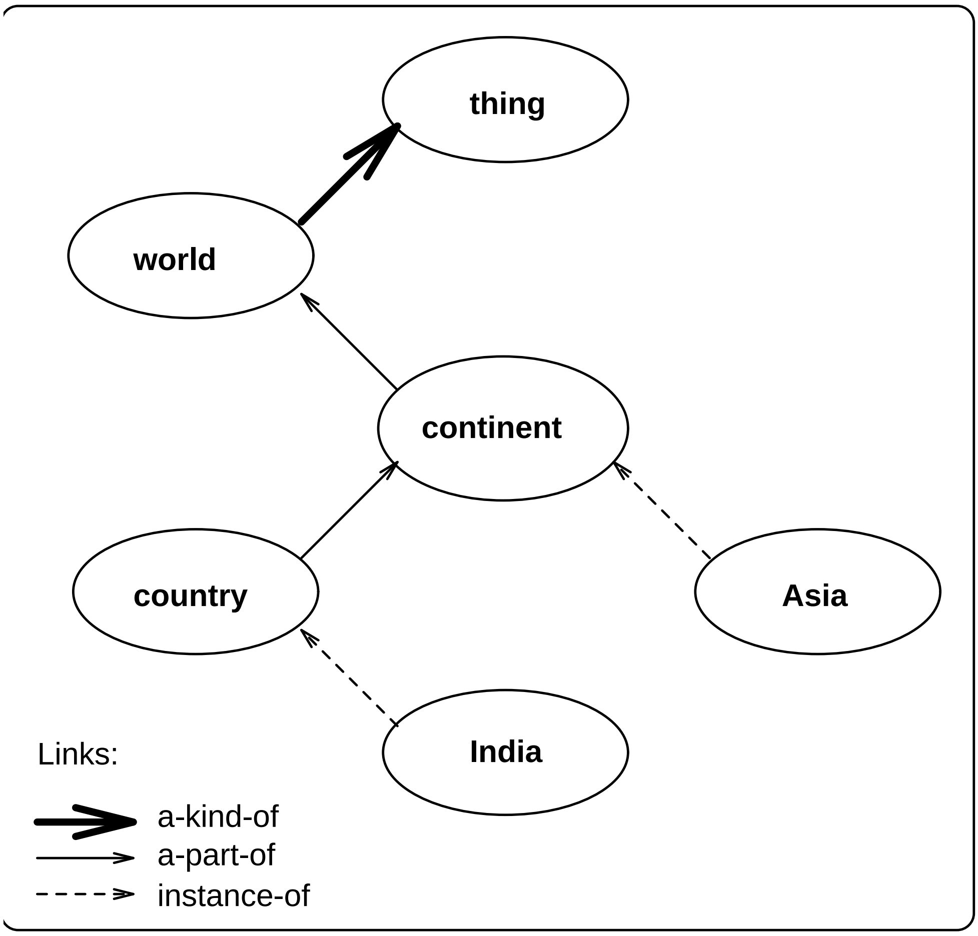

Figure 4.4: Top-level View of Geography Frame System

defined in a particular frame at a particular level in a frame system are inherited (become accessible) by all its descendant frames (either class frames or instance frames).

Inheritance hierarchies are typically set up by **a-kind-of** and **is-a** links. The **part-of** link is another important link for property inheritance.

In the geography frame system (Figures 4.3a and 4.3b), all three types of links are used. In principle, all properties of continents are inherited by the frames for countries. Similarly, the properties of a generic country are inherited by a specific country such as India.

Sometimes, it is useful to prevent such automatic inheritance. The area of a continent is a property of continent. But it is not a parameter that can be inherited by the countries in that continent. Parameters such as number-of-political units, area, perimeter etc. are made specific to certain levels. These parameters are "blocked" from being inherited by nodes lower down, using a "do-not-inherit" attribute; attributes marked as "do-not-inherit" are not inherited by the current frame and its descendants. This is also referred to as *masking*, since attributes are being masked by this method.

As in semantic networks, a frame may have more than one supertype. In this case, the child frame may inherit information from all its supertypes, leading to *multiple inheritance*. For example, in Figure 4.3, we have defined India to be an instance of country, and so India will inherit all the properties of country. But if we had also declared India to be a part of Asia, we would have a situation where India inherits from the frame for Asia as well. Multiple inheritance could occur with the use of any of the standard links – **is-a (instance-of), a-kind-of** and **a-part-of**. There may be problems of consistency when multiple inheritance is permitted, and all these links are considered together. Implementors of frame systems have to keep this in mind, and provide appropriate methods to unambiguously resolve problems that may occur with multiple inheritance.

Constraints on Values

Each slot can specify conditions that its values must satisfy. For example, we may have a condition on a slot for Age, constraining it to be a number, or, more specifically, a number between 0 and 120, say. This will prevent invalid numbers such as 423 being entered as a value for Age. We may also have constraints on the type of the value, on the range of values, or a complex condition involving several such constraints. There could be constraints which specify that a certain slot must only be filled by a frame of a certain type.

Some slots may have multiple correct values. The `number-of-borders` of a country varies, ranging from 1 to (perhaps even) 10 values. We can mark this as `:min 1 :max 10` indicating that we must have at least one value, and upto ten values for that slot. Note that `min` and `max` are not constraints on the *value* of the attribute `number-of-borders`; they specify only the *number of possible values* that this parameter can take.

To define the range of values that a parameter can take, we may define sub-ranges for each such range specification. Alternatively, we may specify the range using predicates such as in-range(X,Y) or member-of(S). Here the constraint on the value is specified by the predicate invoked to test it.

Default Values

Slots may also have default values associated with them. Thus, we could have a frame for a *Car*, which specifies that, by default, it has four wheels. So, if no other information is provided, we understand from this frame that all instances of *Car* have four wheels. These default values can be overridden, if required. Thus, we can still describe a three-wheeled car as a particular instance of the *Car* class, by redefining the wheels parameter at that level as having the value three.

Expectations

Default values in frames, coupled with constraints, define expectations. Even if the actual value of a slot is not known, the constraints and/or the default value provide a partial picture of what is expected. Thus, if a slot is specified as:

Model : (**default** black), or as
Model : (**member-of** (black, blue, red, green))

we can guess that this slot cannot be filled with a completely random value (say, Bombay) – clearly, colour words are expected in these slots. Thus frames also delineate possible values of attributes. This makes the task of representation much easier.

Procedures in Frames

It is not necessary that all the slots have some pre-defined values (such as a number, another frame, a list of names etc.). Some information may be stored as a formula, which has to be evaluated. In some implementations, the user may be provided with *demons* which are triggered by certain conditions. These demons lead to certain expressions being evaluated to provide additional information. (See Section 4.6.2 for details of a particular implementation.)

For example, we may wish to represent a rectangle by its length and breadth. However, we may require other attributes of the rectangle, such as the area or perimeter of the rectangle, as parameters. We can define these as attributes whose values are dependent on the attributes length and breadth. The exact dependency can be specified as a procedure or as a formula (or expression), which may be evaluated when the value of the attribute is required by the system.

For example, we could attach an expression to the slot *area* of a rectangle, thus:

area: (**if-needed** (length $*$ breadth))

This feature provides enormous power to frames.

An *if-needed* procedure attached to a slot tells you how to compute its values if no value has been specified. Sometimes, you need a specific set of actions to be taken as soon as a value is defined for a parameter, or if a value is erased. An *if-added* (or *if-erased*) demon, attached to a parameter, defines a set of actions to be carried out if a value for the attribute is added (or erased). Similarly, an *if-accessed* demon can be triggered if the value of a parameter is accessed.

Multiple Viewpoints

A *frame* can be used to represent a particular instance of an event (present a snap-shot, as it were). A *frame system* can be used to represent the state of the world at different times during the event. This can be done by using a frame corresponding to each point in time that you wish to represent, and building up a montage of these snap-shot frames to provide a complete picture of the event. It is meaningless, in this resulting frame system, to replicate all slots and their values. Values which are common across frames can be shared (using, for instance, some pointer-like mechanism). Only those values that change need be represented anew. Thus frames offer an economical method of representing events.

In [Minsky, 1975;1981], frames were seen as a mechanism supporting multiple viewpoints. Consider two walls A and B, both perpendicular to a given wall C. We can have the frames for walls A and B marked as **perpendicular-to** C. If now we take a right turn in the model, one of these walls (say A) becomes the wall under focus; the other two walls (B and C) become adjacent walls. Minsky showed how these transformations could be efficiently coded in frames, so that frames offered an efficient mechanism to represent multiple viewpoints of the same objects.

In Section 4.6, we will see another method of providing multiple viewpoints.

4.4.3 Identifying Situations and Objects

In the previous section, we have seen various properties of frames and frame systems. The major question to ask now is: How can frames be used?

One simple answer to this is that frames can be used as sophisticated data structures to represent information in an elegant and economical fashion. The application which uses frames would explicitly create or instantiate frames to represent new information. Simple retrieval mechanisms would permit partial match of frames; thus, an application can provide a partial description of a frame and retrieve all matching frames.

It may happen that frames may not have exact matches and we may need to locate the nearest (or approximate) matching frames. This may be required to identify a new situation or a new object, and classify it into one of a known set of prototypical objects or events. While the process of fitting in the event or object may not be precise, it will point out ways in which the new object is different from the prototype. This will, in turn, lead to the generalization of the relevant frames, or the creation of new class frames.

We outline here the use of frames in the identification of situations. The procedure outline is equally applicable to the identification of objects.

Given a situation, to identify it, we select a candidate frame based on some heuristics. If more than, say, two attributes match, then this candidate frame is instantiated. Values are assigned to the attributes, wherever available. Attributes which do not have a value in the new situation may take the default value from the candidate frame.

If a matching candidate frame is not detected, we try again with a small relaxation in one or more of the constraints. Instead of looking for exactly the same frame, we may decide to look for a frame which subsumes the (ideal) target frame. Thus we are looking for more general versions of the same frame. For instance, instead of looking for a *woman* with some properties, we may look for a *person* with similar properties. This relaxation is intuitive and a natural way to search.

Related mechanisms such as scripts [Schank and Abelson, 1977] use similar match procedures. Scripts can be viewed as special purpose frames, describing specific situations. We can have scripts for a birthday party, or for the ritual of eating out at a restaurant. In matching a real situation to the script for the situation, events that are not mentioned are predicted assuming a normal sequence of sub-events. A coherent interpretation of the situation becomes possible, since the script is used with available information to understand the situation. Scripts can be modified on the basis of unusual or unexpected ideas, or new scripts can be created if the unexpected ideas are really different.

4.4.4 Uses of Frame Systems: Centaur

Frames have been a popular knowledge representation formalism, used in a variety of AI programs. For example, ScreenTalk [Chandrasekar and Ramani, 1989] uses frames to create structured representations of the content of natural language sentences.

Many expert system shells support frames as one of the KR formalisms. Frames have also been used in conjunction with other formalisms, such as production rules.

As one major example of the use of frames, we will consider the expert system Centaur [Aikins, 1983]. Centaur was used to reimplement PUFF, a consultation program for pulmonary disease.

One basic idea in Centaur is that frame-like structures may be used to represent rules along with the context in which these rules are applicable. Thus knowledge about control of reasoning can be separated from knowledge about what facts may be inferred given a basic set of facts.

Centaur uses three types of frame-like structures: *prototypes, components* and *facts*. Prototypes are organized into a hierarchy. The top-level Consultation prototype controls the way consultation proceeds, including the stages of initial data entry and hypotheses triggering. The next level consist of prototypes representing different pathological states, such as Obstructive Airways Disease (OAD). The next layer of prototypes represents subtypes of diseases (Asthma, Bronchitis etc.), and degrees of severity of the disease (Mild, Severe etc.). Prototypes may also contain information about prototypes to be instantiated next, and prototypes to be explored next. Information about conclusions to be printed is also stored in prototypes.

There are a number of component-slots in each prototype, which are linked to frame-like structures. A component may describe, for example, a test to be carried out. Each component may contain a special "inference rule" slot, containing a set of rules to be invoked to find the value of the component, failing which the system prompts the user for the value. Centaur provides a special language for writing rules.

Prototypes may also contain meta-level knowledge about reasoning with these structures in *control slots*.

Fact-frames are associated with every domain parameter. Each fact frame has six slots, namely: the name of the parameter, its value, its degree of certainty, its source, classification and its justification.

Centaur works by using the initial data (got by triggering the consultation prototype) to trigger prototype instantiation. Then one prototype, representing some pathological state, is selected as a candidate for further exploration. Using known facts to fill the prototype, Centaur compares expected values of parameters with actual values, and uses the discrepancy to refine the diagnosis. Thus the prototypes act as a repository of facts and rules which help to compare expected values of parameters in a given situation with actual parameters encountered.

To summarize, frames are used in a variety of ways in Centaur. Frames are used as a representation of prototypical knowledge. Rules are embedded as one kind of slot fillers, in these frames. Contextual knowledge is provided along with the frames, to guide the reasoning process. At the basic level, frames are used in facts as structuring devices, to store a set of related attributes and their values.

4.5 Semantic Network and Frame Languages

Because of their simple structure, several languages have used semantic networks to represent knowledge, such as the SNePS Semantic Network Processing System [Shapiro, 1979]. SNePS was one of the first knowledge representation languages to have the expressive power of first-order logic.

Conceptual graphs [Sowa, 1984] can be viewed as special purpose semantic networks, with clearly specified links and a clear semantics.

Winograd, in his paper *Frame Representations and the Declarative Procedural Controversy* [Winograd, 1975], tries to synthesise procedures and declarative structures in a frame-like system. This work was a precursor to the frame language KRL. KRL [Bobrow and Winograd, 1977] was meant to be "a [KR] language that will integrate procedural knowledge with a richly structured declarative representation designed to combine logical adequacy with a concern for issues of memory structure and recognition-based reasoning processes." KRL had interesting notions of how and why procedural attachment should be supported. However, it was not a very popular or successful language, even though it has pedagogic value.

[Hayes, 1979] was the first serious attempt to analyse the meaning of frames. In his conclusion, Hayes comments that: "... we could sum up *frames* as the suggestion that we should *store* assertions in nameable 'bundles' which can be retrieved via some kind of indexing mechanism on their names. In fact, the suggestion is that we should store assertions in non-clausal form."

The KL-ONE formalism [Brachman and Schmolze, 1985] and the FRL system [Roberts and Goldstein, 1977] are other important landmarks in the history of frame systems.

FrameKit [Nyberg, 1988] is a frame-based knowledge representation system written in Common LISP, which provides the basic mechanisms of frames, inheritance, demons and the concept of views. In the next section, we will examine this language in some detail.

[Finin, 1986] describes a simple frame based representation language called PFL (a Pedagogic Frame Language). The entire code for PFL (which is also written in Common LISP) is available in the technical report cited.

There have been many hybrid KR schemes. [Brachman et al, 1983] describes KRYPTON, a KR system with two main components: a terminological component, where a frame-like language is used to describe terms, and an assertional component, where a first order predicate language is used to make assertions about these terms. Both these languages are embedded in a functional framework, so that the user is insulated from the structures that implement the terminological and assertional components.

Today, many AI and expert systems tools provide a variety of knowledge representation tools. The simpler ones provide just one or two representations, while higher end tools such as ART and KEE provide several formalisms (rules, logic, frames, objects etc.).

4.6 An Overview of FrameKit *

FrameKit was first developed by Jaime Carbonell at the Carnegie Mellon University (CMU). The current version is described in [Nyberg, 1988], and the overview presented in this section is based on that description.[4] Several applications have been developed using FrameKit, including the Knowledge Based Machine Translation system built at the Center for Machine Translation at CMU.

In FrameKit, frames are abstract data types comprised of slots, facets, views and fillers. Each frame can have any number of slots, and each slot may have any number of facets. In turn, each facet can have any number of views and each view may have any number of fillers. Slots, facets and views can be added and deleted at run time.

The following Common LISP command shows you how to create an example frame in FrameKit. Commands to Common LISP are shown, and the value returned by the LISP system is given after '=>'.

```
  (make-frame  NCST
     (is-a (value (common research-lab
                          not-for-profit-institution)))
     (Location   (City (common Bombay))
                 (Country (common India)))
  )
=> NCST
```

Here, *NCST* is the name of the frame, with two slots is-a and Location. The is-a slot has a single (predefined) facet Value, with two fillers: *Research Lab* and *Not-for-Profit-Institution.* The Location slot has two facets, City and Country, with one filler each. All these are stored in the view called common. A later section (Section 4.6.5) discusses views and the view mechanism in FrameKit.

4.6.1 The Structure of Frames in FrameKit

Each frame in FrameKit is a simple nested list structure. Here is a BNF definition of a frame:

* This section may be omitted during a first reading.

[4]Since FrameKit uses LISP to manipulate frame structures, this section will include some Common LISP code. Thus the reader may need to have some knowledge of Common LISP to fully appreciate the potential of FrameKit.

```
<frame> ::= (<name> <slot>*)
<slot>  ::= (<name> <facet>*)
<facet> ::= (<name> <view>*) | <predefined-facet>
<view>  ::= (<name> <filler>*)
<filler>::= <value>* | <demon>* | <predicate>*
<predefined-facet>  ::= (VALUE <view>*)        |
                        (IF-ADDED <view>*)     |
                        (IF-NEEDED <view>*)    |
                        (IF-ERASED <view>*)    |
                        (IF-ACCESSED <view>*)  |
                        (RESTRICTIONS <view>*) |
                        (DEFAULT <view>*)
```

Here <name> is any Common LISP symbol and <value> is any Common LISP expression (usually a symbol or a number). <demon> is a Common LISP expression, generally a call to a function. <predicate> is a Common LISP expression which evaluates to Nil or non-Nil, corresponding to False or True.

The predefined facets handle demons and inheritance. The views in Value and Default facets have fillers of type <value>, while If-Added, If-Needed, If-Erased and If-Accessed facets have fillers of type <demon>. The views in Restrictions facets have fillers which are of type <predicate>.

4.6.2 Demons in FrameKit

Demons have access to the run-time context through the special variables !Frame, !Slot, !Facet and !Filler, which contain the name of the current frame, slot, facet and filler respectively. When a filler is added to the Value facet of a slot, the slot is checked to see if it has an If-Added facet. If it does, the fillers (demons) attached to the If-Added facet are evaluated as Common LISP expressions, leading to (planned) side-effects.

Consider the example below:

```
  (make-frame square
     (side (value) (if-added '(update-values !frame !filler)))
     (area (value))
     (perimeter (value)))
=> SQUARE
```

with the associated Common LISP function:

```
  (defun update-values (square side)
     (replace-value square 'area      (* side side))
     (replace-value square 'perimeter (*  4   side)))
=> UPDATE-VALUES
```

If we now create a square and assign the value 5 to the slot side, using:

```
  (add-value 'square 'side 5 :demons t)
=> T
```

we will find that values of `area` and `perimeter` are updated:

```
  (get-values 'square 'area)
=> (25)

  (get-values 'square 'perimeter)
=> (20)
```

Similarly, the If-Needed demons are triggered if some value(s) are required, and these are currently set to NIL; the associated Common LISP expressions provide the value(s) required. In the example above, we could have used an If-Needed demon attached to `area` (or `perimeter`) instead of an If-Added demon attached to `side`. This would be triggered when the value of `area` is required – the appropriate Common LISP expression would then obtain the value of `side` from the frame, and evaluate `area`.

The If-Erased is similar to If-Added and is triggered when some value is removed. The If-Accessed is similar to If-Needed, except that it is fired whenever the associated value is accessed.

4.6.3 Relations in FrameKit

FrameKit lets the user define relationships between frames. We say a link exists between two frames if there is a facet in one that contains the other as a filler. A relationship is a two-way link between frames.

There are three pairs of predefined relations in FrameKit, Is-A/ Subclasses, Instance-Of/ Instances and Part-of/ Parts, where each relation has been paired with its inverse relation.

You can define frames such that if a link is created, its inverse link is automatically defined by FrameKit. This can be a powerful tool in describing and maintaining representations.

4.6.4 Inheritance, Restrictions and Default Values

FrameKit supports the standard inheritance mechanism. It also supports multiple inheritance (inheritance through multiple links). If a slot has a value locally, inheritance is not invoked. FrameKit traverses Is-A and Instance-Of links during inheritance. The mechanism used in determining inheritance can also be controlled by the user.

The example given below should be instructive. `:inherit t` in the last statement forces inheritance of fillers. By default, `get-values` gets only the values defined locally.

```
  (make-frame car
      (wheels (value (common 4))))
=> CAR

  (make-frame maruti
      (instance-of (value (common car))))
```

```
=> MARUTI

  (get-values 'maruti 'wheels :inherit t)
=> (4)
```

FrameKit also supports the notion of restrictions on fillers. We can use some Common LISP code to add a filler to the `car` frame defined above, to restrict the number of wheels to be even:

```
  (add-filler 'car 'wheels 'restrictions '(evenp !filler))
=> T
```

Then a call such as the following (to set the value of wheels to 3) will result in an error message.

```
  (add-value 'maruti 'wheels 3 :demons t)
=> FrameKit> ADD-FILLER: failed restriction on filler 3
   FrameKit>             in view COMMON facet VALUE
   FrameKit>             slot WHEELS frame MARUTI
  NIL
```

while the following call (to set the value of wheels to 4) succeeds:

```
  (add-value 'maruti 'wheels 4 :demons t)
=> T
```

FrameKit also allows default values to be represented, as shown in the example below:

```
  (add-value 'car 'wheels 'default 4)
=> T

  (make-frame premier
     (instance-of (value (common car))))
=> PREMIER

  (get-values 'premier 'wheels :inherit 'default)
=> (4)
```

4.6.5 Views in FrameKit

FrameKit's View mechanism allows users to maintain multiple representations of a given frame within the same knowledge base. This is equivalent to having different views of the same frame, for use in different contexts. If a particular facet in a frame contains fillers from different representations of the frame, these fillers may be differentiated using the View mechanism of FrameKit.

For example, you may have a collection of addresses to be represented. In displaying entries from that database, you may wish to show foreigners the country code in a phone number. You may not want this information to be shown to local people. You can achieve this as shown in the following definition and queries.

```
  (make-frame address-record
              (name (value))
              (phonenumber (value (foreign '+91-22-620-1606)
                                  (common  '022-620-1606))))
=> ADDRESS-RECORD

  (get-values 'address-record 'phonenumber :view 'foreign)
=> (+91-22-620-1606)

  (get-values 'address-record 'phonenumber)
=> (022-620-1606)

  (get-values 'address-record 'phonenumber :view 'all)
=> (+91-22-620-1606 022-620-1606)
```

The default view is the common view. All the views can be accessed using the `:view 'all` keyword, while a particular view can be got using the name of that view.

This mechanism, which is not very common in frame languages, can be quite useful in a representation task.

4.6.6 FrameKit: Summary

In this section, we have tried to show, very briefly, the main features of FrameKit. Clearly, this is not a complete description. For further details, the user may refer to [Nyberg, 1988]. [5]

We have chosen to describe FrameKit in this section because it is reasonably complete and self-contained, and because there have been practical systems built on top of this package. However, there are several other frame systems. Some have remained in the laboratories, while others have been used at least in one system.

4.7 Conclusions

In this chapter, we first examined the mechanism of associative triples. We then examined the structure and utility of semantic networks and their drawbacks. We then examined frames and frame systems, and their properties. We briefly considered how frames were used in AI systems. We also discussed a particular frame language in some detail.

Levesque and Brachman suggest that there is a tradeoff between the expressiveness and tractability (or computational power in determining the truth conditions of sentences in that representation) of a knowledge representation formalism [Levesque and Brachman, 1984]. They look at different formalisms and evaluate them in terms of what they can be used to express, and the reasoning power that they require. They suggest that KR formalisms be designed with this expressiveness-tractability tradeoff kept in mind.

[5]FrameKit is available as public domain code. Contact Eric.Nyberg@cs.cmu.edu by electronic mail for details about how to obtain this system.

We have to achieve a balance between expressiveness of a representation and the complexity of computing truths in that representation. There is no single "best" representation for any AI program. For most applications, we may need a mixture of representations to satisfy our representation needs.

Summary

- There is a need for structured formalisms since logic and rule based systems do not have simple mechanisms for organization of knowledge.
- Triples are simple object-attribute-value representations; they can be used to model simple chunks of information.
- Semantic networks consist of nodes connected by different kinds of associative links. Nodes can be type nodes or token nodes.
- Search in semantic networks can be done using spreading activation.
- Semantic networks use the notion of property inheritance where properties are inherited from type-to-type or from type-to-token across certain kinds of links.
- Semantic networks have been criticised for using ad hoc representations, arbitrary links and unclear semantics.
- Frames are general purpose clusters of facts. Frames define prototypical situations or objects. A frame system is a collection of (related) frames.
- Frames can be used to represent default or constraint information, provide multiple perspectives and permit procedures to be stored as values.
- Inheritance, subsumption and disjointness are important concepts in frames.
- There are several semantic network and frame based languages, most of them developed in research environments.
- The representation one chooses for a problem is intricately connected to the nature of the problem. Hybrid representations might turn out to be necessary for some applications.
- FrameKit is a particular frame-based knowledge representation system which has been used in several applications.

Exercises

1. Give two examples of type-token relationships. Why are they so popular in Knowledge Representation systems?
2. Consider the following information:

(i) Mammals are warm-blooded.
(ii) Mammals have 4 legs.
(iii) Tigers eat meat.
(iv) Tigers are mammals.
(v) Tigers are dangerous.
(vi) Hobbs is a tiger.
(vii) Hobbs is not dangerous.
(viii) Raja is a tiger.
(ix) Raja has three legs.

Construct an inheritance hierarchy with this information, along with exceptions (where necessary). Using this hierarchy, answer the following.

(a) Which are the dangerous tigers?
(b) How many legs does Raja have?
(c) How many links have to be traversed in the hierarchy to check if Hobbs is warm-blooded?
(d) Are all mammals dangerous?
(e) What facts can you deduce about Raja from the hierarchy?

3. Geometrical figures such as polygons, rectangles, etc. have an important property. Most of them are special cases of a more general figure, as explained in the following paragraph.

> A quadrilateral is a polygon with 4 sides. A parallelogram is a quadrilateral in which opposite sides are parallel. If you specialise this further by restricting the four angles to be right angles you have a rectangle. A rectangle becomes a square if you make all sides equal.

Represent this hierarchy of mathematical figures using a frame structure. Show an example of inheritance with this representation. You can start as follows:

Frame Quadrilateral **ako** Polygon
  number_of_sides : 4
  perimeter : ...
end Frame

Frame Parallelogram **ako** ...

An optional exercise: Notice that the two specialisations mentioned above can be done in the reverse order generating a rhombus (a parallelogram with all sides equal) as the intermediary instead of a rectangle. How will you represent this?

5. Reasoning under Uncertainty

5.1 Introduction

Uncertainty is inherently part of most human decision making. Expert systems which seek to model human expertise in specific domains have to take care of uncertain information. They, therefore, need methods to represent uncertainty in knowledge and data and should be able to reason with such uncertain information.

In this chapter, we examine two of the commonly used approaches to handling uncertainty in expert systems, namely, probability theory and certainty factor method. Dempster-Shafer theory and fuzzy logic are two formal approaches for uncertainty reasoning being adapted to the domain of expert systems. We will examine these two approaches in Chapter 10.

5.2 Sources of Uncertainty

In an expert system application, uncertainty can arise from various sources.

One of the primary causes of uncertainty is that the domain knowledge used is not always reliable. Expert systems typically use shallow reasoning using thumb rules or heuristics provided by the expert. These rules are not guaranteed to be correct always. To a question about correctness of a domain heuristic, the usual response from experts will be that "it works most of the time". In other words, the implications in the rules are weak. Such pieces of heuristic knowledge will hold in most of the cases (say, 90% of the cases).

Thus the standard form of an if-then rule, **If A,B,C then D**, is not usually sufficient to represent domain knowledge. Rules will have to take the more general form: **If A,B, C then D with certainty X** where the quantity X represents the degree of belief or confidence in the rule.

Inaccuracy of instruments used to measure parameter values is another source of uncertainty. Often, values of a parameter required by the system are input by a user or measured by sensors. Inaccuracy in these leads to uncertainty in the data obtained, which in turn affects the conclusions. For example, consider the question "Do you have chest-pain?". Given the observation that, occurrences of chest pain can vary in intensity (from very mild to very severe) as well as in frequency (occasional to almost continuous), a simple yes-or-no answer will be difficult to comprehend. A patient may even answer "I think so". This answer has to be treated as a *partial* yes.

Another cause of uncertainty is the inherent imprecision of the language in which the information is conveyed. Questions such as *Is the person tall?, Is the weather warm?*, etc. cannot, in general, be answered with full certainty. This is partly because of the difficulty in defining precisely the meaning of words such as *tall* and *warm*. For example, a person whose height is 6 feet is tall, but what about someone who is 5 feet and 10 inches? 15 degrees celsius may be warm in the winter, but not in the summer!

Uncertainty can also arise because of incomplete knowledge. In most complex decision making processes, all the required information may not be accessible when required. Decisions have to be taken without knowing all the relevant parameters. For example, a doctor cannot wait till all test results are available to treat a patient complaining of severe abdominal pain. Experts, and hence expert systems, will have to make at least tentative decisions based on available (partial) information. The system should be willing to retract such decisions, if, later on, more evidence is obtained against them.

There is another source of uncertainty in systems which deal with multiple knowledge sources. There may be different experts with expertise on different aspects of the problem. Occasionally, they may arrive at conclusions that are not consistent with one another.

Similar problems arise with multiple sensors measuring related quantities. The inputs from all these sources have to be combined. There are essentially three types of problems that can arise from such combination: conflict, subsumption and redundancy. Conflict arises when the conclusions from two sources are inconsistent with each other. When both give the same information, we have redundancy. If the parameters used by both the sources are the same (or have a significant overlap), we may end up attributing more confidence to the result by counting the parameters twice. When the conclusions of one source are a special case of the conclusions of another, we have subsumption. For more information about common problems in rule bases, refer to Chapter 9.

5.3 Approaches to Handling Uncertainty

Approaches to handling uncertainty vary from system to system, depending on the type and cause of uncertainty in the domain of application. These approaches can be divided into two broad categories based on their characteristics: those representing uncertainty using numerical quantities and those using symbolic methods. There are pros and cons for both approaches. Most expert systems today provide some support for handling uncertain information. Most of them are extensions or variations on the basic approaches that we will be considering in this chapter. There is no single universal method yet to handle all forms of uncertainty. Considering the complexity of the various forms of uncertainty, it is unlikely that there will ever be one!

In numerical approaches, one models the uncertainty in data and knowledge by numbers and provides some algebraic formulae to propagate these uncertainty values to the conclusions. These approaches have been widely used in expert system applications. Some examples of numerical approaches are Bayesian reasoning, Confirmation theory [Shortliffe and Buchanan, 1975], Evidence theory [Gordon and Shortliffe, 1984] [Shafer, 1976] [Zadeh, 1984;1986] [Cortes-Rello and Golshani, 1990] and Fuzzy set approaches [Negoita, 1985] [Gordon and Shortliffe, 1985]. Numerical approaches are suitable for addressing problems related to *unreliable* or inaccurate knowledge. But they normally run into problems in providing a natural representation for uncertainty due to *incomplete* information.

On the other hand, symbolic characterization of uncertainty is mostly aimed at handling incomplete information. These approaches have been found to be unsuitable for handling other types of uncertainty. Examples of symbolic approaches are Assumption Based Reasoning [Doyle, 1979] [deKleer, 1986], Default Reasoning [Reiter, 1980], Non-monotonic Logic [McDermott and Doyle, 1980] and Circumscription [McCarthy, 1980]. Commercially available tools such as ART, KEE, etc. support Assumption Based Reasoning.

Hardly any of these approaches are concerned with uncertainty arising out of multiple knowledge sources or multiple sensor reports. We will also ignore this problem. In this chapter, we will concentrate on numerical approaches for handling uncertainty. We will discuss the probabilistic method of Bayesian reasoning and Confirmation theory used in MYCIN. In Chapter 10, we will briefly discuss symbolic uncertainty handling.

5.4 Probability Theory

The mathematical view of probability is based on the relative frequency of occurrence of some events contingent on some other events. For details on probability theory, readers are referred to any textbook on probability or statistics. We will, however, review some concepts essential for our purpose.

5.4.1 Conditional Probability

The probability of occurrence of an event is called the unconditional probability or the *a priori* probability of the event. The probability of occurrence of an event can change as you get to know more and more about the environment, such as the occurrence of other events.

For instance, consider a system for diagnosing diseases. If 1% of the people on the average have been found to have disease X, then there is an *a priori* probability of 0.01 that any given person has disease X.

However, if you consider only the people showing a particular symptom S, the percentage of people having disease X may be higher, say 20%. Thus, knowing that the particular person exhibits a symptom S, leads you to suspect that he has disease X with a higher probability (0.2). The probability of some event E, given that another event E1 has already occurred, is known as the conditional probability of E given E1, denoted by P(E | E1). This is essentially the unconditional probability of event E obtained by restricting your world to only those individuals who satisfy event E1.

This seems to provide a way to represent uncertainty in the context of expert systems. Consider the task of a diagnostic expert system. Let S be the set of all symptoms in the domain and H be the set of hypotheses (i.e., possible diseases). Assume that we have all the conditional probabilities P(h | s) for each h in H and each possible subset s of S [1]. Given a particular case to diagnose, one can identify the set of symptoms s (where s is a subset of S) present in that particular case. The diagnosis will be a permutation of the elements of H, in the order of conditional probability of occurrence given the symptoms s. For each hypothesis h of H, the probability that h occurs in this case is P(h |s). This can be picked up from the probability data collected.

5.4.2 Problems with Conditional Probability Approach

This approach, though mathematically sound, is not easy to use. First of all, getting all the data required for any application will not be easy. If H contains N_h elements and S contains N_s elements one requires at least $N_h * 2^{N_s}$ probability values, where 2^{N_s} is the number of possible subsets of S (in other words, the different combinations of symptoms that can arise). If N_s is 10, and N_h is 10 (a very trivial problem), the number of probability values required is 10240. In a real situation, N_s could be several hundreds and then the number of values required becomes extremely large.

Another problem with this approach is getting the particular types of data required. We need figures such as the percentage of people who suffer from jaundice given that they have yellow eyes. For this, one has to find out how many people have jaundice out of the total number of people who have yellow eyes. Obtaining valid data for all such combinations is practically impossible in most cases. Hence one tends to rely on estimates of probabilities. Normally, doctors find it difficult to provide data such as the number of people suffering from jaundice out of all the people complaining of yellow eyes. However, the inverse of this data might be easier to obtain. The number of people who have yellow eyes out of those suffering from jaundice can be obtained from the medical records of a hospital.

We can use Bayes' theorem to deduce the probability of jaundice given yellow eyes from the probability of yellow eyes given jaundice. Bayes' theorem states that

$$P(h|s) = (P(s|h) \times P(h))/P(s)$$

For example, let

$P(j\|y)$	be the probability of a person having jaundice, given that he has yellow eyes,
$P(y\|j)$	be the probability of a person having yellow eyes given that he has jaundice
$P(j)$	be the probability of a person having jaundice and
$P(y)$	be the probability of a person having yellow eyes

then $P(j|y) = (P(y|j) \times P(j))/P(y)$

For example, analysis of the records in a hospital provides information such as the following:

---

[1]Each person may exhibit only some symptoms. Hence a person is represented by a subset s of S, which indicates the symptoms that he exhibits.

Number of patients examined N	1200
Patients with yellow coloured eyes N_y	40
Patients found to have jaundice N_j	10
Patients with yellow eyes and found to have jaundice N_{jy}	8

Then, we can compute:

Probability of Jaundice P(J)	$= N_j/N$	$= 10/1200$
Probability of Yellow eyes P(Y)	$= N_y/N$	$= 40/1200$
Probability of Yellow eyes given Jaundice P(Y|J)	$= N_{jy}/N_j$	$= 8/10$

Now using Bayes' theorem, we can compute the probability of Jaundice given yellow eyes as $((8/10) \times (10/1200))/(40/1200)$, i.e., 0.2.

However, this does not solve the earlier problem; a large number of values are still required. We need as much data as we needed for the earlier approach plus the values of P(s) for all subsets s of S.

One way to reduce the amount of data required is to assume that symptoms are independent of each other. Consider two symptoms, fever and yellow eyes. In general, the chances of one having fever may be higher if one has yellow eyes. Thus, knowing the probability of fever and that of yellow eyes, one cannot predict the probability of (fever `and` yellow eyes) because they are not independent of each other. In general, we have

$$P(A\, and\, B) = P(A|B) * P(B)$$

Therefore P(A and B) cannot be deduced from P(A) and P(B) alone. Thus in the general case, we have to collect the joint probabilities of every combination of symptoms as well as the individual probabilities of the various symptoms. Under the assumption of independence of symptoms, the probability of A is *not* affected by the occurrence of B. Hence $P(A|B) = P(A)$. Thus knowing P(A) and P(B) one can compute P(A *and* B), without requiring the conditional probabilities of the A-B pair. Thus, in the formula giving the total number of probability values, viz. $N_h * 2^{N_s}$, the factor 2^{N_s} may be substituted by N_s, since the probabilities of the combinations can be computed from the basic probabilities using this method.

This significantly reduces the number of values required by the Bayesian reasoning procedure. However, the assumption of independence required to achieve this reduction is not valid in most cases and may lead to erroneous results.

Yet another problem with uncertainty reasoning based on probability is the non-intuitiveness of the approach. Human experts do not use such figures in their reasoning tasks. Moreover, the relations between various symptoms and hypotheses are hidden in the probabilities used in the representation. This makes it difficult to provide any reasonable explanation about the behaviour of the system to the users. The domain relations and associations being hidden also make it difficult to debug these systems.

Another problem with the probability theory approach is the following. Given the conditional probability, P(H|E1), the theory requires that $P((not\, H)|E1) = 1 - P(H|E1)$. That is, partial evidence for a hypothesis is also construed as partial evidence for the negation of the same hypothesis. Experts are not normally comfortable with this notion. When they say that the presence of fever indicates malaria with probability 0.3, they do not mean that fever is an evidence against malaria with probability 0.7.

5.4.3 The PROSPECTOR Approach

PROSPECTOR [Duda et al, 1978], one of the early expert systems in the domain of mineral exploration, makes use of a probability based approach to handle uncertainty. PROSPECTOR is described in some detail in Chapter 7. The task of the system was to advise geologists about the possibility of finding various ore deposits at a given geographical area. The mechanism is essentially based on Bayesian reasoning. The knowledge base consists of a directed network of facts and hypotheses. For example,

> **If** *Coeval Volcanic Rocks (CVR) are present*
> **it indicates** *Favourable Level of Erosion (FLE).*

This will be represented by connecting the nodes representing CVR and FLE by a directed link.

The inter-connecting links indicate how the truth or falsity of a fact or hypothesis affects the truth or falsity of another. Each link (say from A to B) has two numbers associated with it: a necessity factor (LN) and a sufficiency factor (LS). Each node has an *a priori* probability value associated with it. The updating mechanism works with a form of representing probability called **Odds**. The odds can be derived from the probability, P, using the formula $O = P/(1 - P)$. Thus if the probability P is zero, O is also zero, and if the probability is 1, O is infinity. When the probability associated with a node changes due to some information provided by the user, the system uses the factors, LN and LS, to propagate the changes to all nodes connected to it. When an intermediate node changes its probability, the same mechanism is followed to propagate the change to all higher level nodes connected to it, till the top level nodes are reached.

Given the Odds-before (O_b) for a node N and the values LS and LN of a link connecting a node R to N, the Odds-after (O_a) is obtained as

$O_a = LN \times O_b$, if R is known to be false

$O_a = LS \times O_b$, if R is known to be true

The necessity factor, LN, indicates how necessary R is for N to be true. If this factor is very low (much less than 1) and R is false, the odds of N is reduced significantly. The second factor LS, the sufficiency factor, indicates how sufficient R is for N to be true. If this factor is high (much greater than 1) and R is true, N's odds are increased substantially. In general, R may not be known to be absolutely true or false. In such cases PROSPECTOR uses an interpolation mechanism to combine the updates. The system uses an incremental mechanism of updating the odds on a node, as it gradually gathers information on the odds of nodes connected to it. The overall mechanism followed in PROSPECTOR is similar to that of MYCIN, which we will discuss shortly. MYCIN follows a more heuristic approach to uncertainty propagation, compared to PROSPECTOR which uses a mechanism based on the Bayesian formula. Note that LN and LS are defined using conditional probability and are used commonly in statistics (where they are called likelihood ratios):

$LN = \frac{P(not\ E|H)}{P(not\ E|not\ H)}$

$LS = \frac{P(E|H)}{P(E|not\ H)}$

where, $P(E|H)$ refers to the conditional probability of evidence E given hypothesis H. These LS and LN ratios will be attached to the link from the node labelled E to the node labelled H. More details about this approach (evidence combining formulae etc.) can be found in [Duda et al, 1978].

5.5 Certainty Factors

In general, one prefers a framework based on sound mathematical foundations to an *ad hoc* method. Hence the first attempts at modelling uncertainty were all based on probability theory. However, as mentioned above, probability theory based approaches soon ran into problems that were difficult to overcome. Hence expert systems researchers turned to more heuristic formalisms. There are a number of approaches being followed by various systems. We present the general framework of such approaches, called the certainty factor model, commonly used with if-then rule based systems. As an example of the use of certainty factors, we discuss the specific approach used to model uncertainty in the MYCIN expert system [Buchanan and Shortliffe, 1984].

5.5.1 The General Characterisation

Consider a general if-then rule,

R1: **If** A **and** B **and** C **then** D

There are two types of uncertainty that we have to capture in a system that uses rules of this form for knowledge representation:

1. *The uncertainty in the data used:* To model this, we attach a numerical measure with every input that the user provides the system. This measure, in general, models such aspects as inadequate information, unreliable sensors, ambiguous language, strength of belief in the data, etc.

2. *The uncertainty in the rule:* This reflects the expert's confidence in the particular rule. Another numerical measure will be attached to every rule to represent this. The exact interpretation of this quantity varies from system to system. In MYCIN, this is a single number, interpreted to be the measure of certainty to be associated with the conclusion, *when one is certain that all the antecedents of the rule are true.*

In the general if-then framework, the uncertainty in the data and rules have to be propagated to the conclusions. Consider rule R1 given above. Let U_r be the uncertainty associated with this rule. Some of its antecedents may be based on data directly available from the user. This data will in general, have a measure of uncertainty associated with them. The uncertainty of such an antecedent will be calculated from the uncertainty associated with the data used in the antecedent. The other antecedents will be deducible from rules in the rule base. The process of propagating the uncertainty to intermediate and final conclusions implies that these antecedents will also have a measure of uncertainty attached to them.

We need some formalism to propagate the measures of uncertainty in the antecedents, and the uncertainty attached to the rule, to the conclusion of the rule. This is normally done in two steps.

The different antecedents in the rule will, in general, have different values of uncertainty attached to them (for example, U_1, U_2 and U_3 in Figure 5.1). We need some formula (F1) to combine these measures and give us a consolidated uncertainty figure, U_{123}, of all the antecedents. For example, one can take a safe option and say that one's belief in the antecedents is only as much as that of the least certain antecedent. Another option would be to use the average of the uncertainties of all the antecedents.

Having obtained a measure of uncertainty for the set of antecedents, we need another formula (F2) to combine this measure with the measure of uncertainty attached to the rule (U_r) to give us a measure of uncertainty for the conclusion of the rule. That gives us a measure of uncertainty, U_{R1}, for the conclusion based on the rule R1 that we had been considering. The actual formula varies widely, depending on the meaning/interpretation given to the measure of uncertainty of the rule.

In general, there may be more than one rule to deduce a conclusion. We need some formula (F3) to combine the contributions from the different rules into a single measure, Uconcl. There are various options available here as well. One simple solution would be to take the highest contribution as the net contribution.

5.5.2 The MYCIN Model

The model of inexact reasoning used in MYCIN is an *ad hoc* model which fits very well into the formalism discussed in the previous section. This is an intuitive approach which suits the rule based representation paradigm. The MYCIN approach can be seen to resemble, in some respects, the Bayesian probability approach which we have already discussed and Fuzzy Set theory which we will be discussing in Chapter 10.

The MYCIN approach is based on numerical measures called Certainty Factors (CFs). Sometimes they are also called Confidence Factors. Certainty factors are numbers in the range −1.0 to +1.0. CFs are attached to every data element and every hypothesis, reflecting the current confidence in that piece of data or hypothesis. A value of +1.0 indicates absolute belief and −1.0 indicates absolute disbelief.

In the context of the framework described in the previous section, the formulae characterising the approach are described below and summarised in Figure 5.2.

Given a set of antecedents with associated CFs, the net belief in the antecedents is taken to be the minimum of the CFs of all the antecedents (formula F1).

$$\text{CFantecedents} = \text{minimum of CFs of all antecedents} \qquad \ldots\text{F1}$$

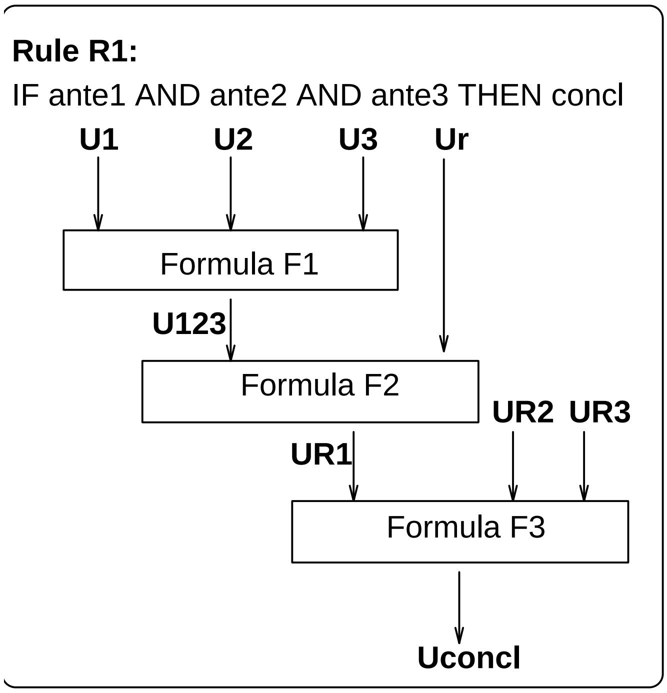

Figure 5.1: The General Framework for Uncertainty Reasoning in Rule Based Systems

Thus MYCIN takes the safe option, by taking the strength of the weakest link in a chain as the strength of the chain.

For combining the certainty factor of the rule and the certainty factor of the antecedents of the rule, MYCIN uses the following formula (formula F2):

CF of conclusion from rule =
CF associated with rule R1 $\times$ CFantecedents ...F2
(provided CFantecedents >= 0.2)

Note that an if-then rule is applicable only when the antecedents of the rule are all true. In the absence of uncertainty, evaluating truth is a straightforward task. With certainty factors as in MYCIN, an antecedent is considered true only if it attains a CF greater than 0.2. The particular threshold of 0.2 used here is essentially an arbitrary choice. A CF value of greater than zero could be interpreted as true. But one wants to suppress or ignore options that do not have significant support so as to avoid working on hypotheses such as "brain tumour", CF = 0.000001. Hence a threshold above zero seems reasonable to use.

Note also that in formula F2, the CF obtained for a conclusion from a particular rule will always be less than or equal to the CF of the rule. This is consistent with the interpretation of the CF used by MYCIN, that is, the CF of a rule is the CF to be associated with the conclusion if all the antecedents of the rule are known to be true with complete certainty. When the antecedents are known to be true with less than full certainty, the CF of the conclusion will be weighted down by the uncertainty of the antecedents.

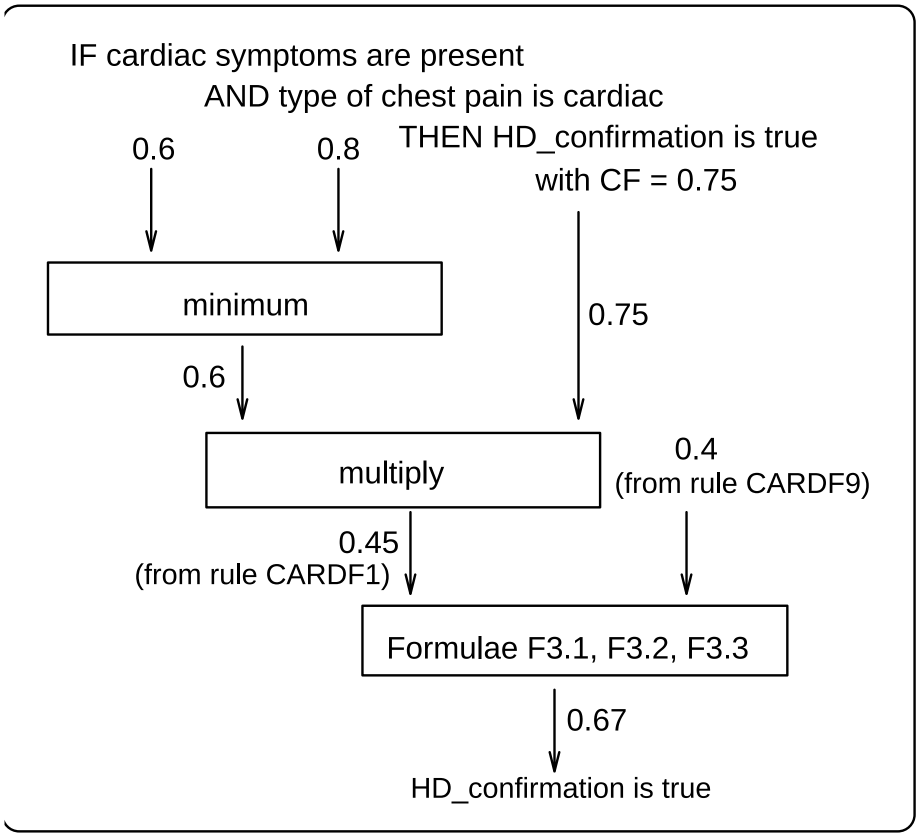

Figure 5.2: An Example of MYCIN CF Computation

In general, there will be more than one rule in the rule base that are applicable for deriving a specific conclusion. Some of them will not contribute any belief to the conclusion (possibly because one or more of their antecedents have a CF less than the threshold of 0.2). The contributions from all the other rules for the same conclusion, have to be combined. The method used by MYCIN (formula F3.1 to F3.3) for this purpose is as follows:

```
CFnew = CFold + CFin * (1 - CFold),            ... F3.1
         when (CFold, CFin > 0)

CFnew = CFold + CFin * (1 + CFold),            ... F3.2
         when (CFold, CFin < 0)

            CFold + CFin
CFnew = -------------------------  , otherwise. ... F3.3
         1 - min (|CFold|, |CFin|)
```

To start with, assume that the CF of a conclusion is 0.0 (that is, there is no evidence in favour or against). As different rules for the conclusion fire, this CF gets updated. MYCIN uses a method that incrementally updates the CF of the conclusion as more evidence is obtained. Let CFold be the CF of the conclusion so far, say after rules R1, R2, . . . Rm have been fired. Let CFin be the CF obtained from another rule Rn. The new CF of the conclusion (from rules R1, R2, ..., Rm and Rn), CFnew, is obtained using the formulae given above.

5.5.3 An Example

To clarify these concepts, consider the following rules:

Rule cf1:

> **If** cardiac symptoms are present
> **and** the type of chest pain is cardiac
> **then** HD_confirmation is true with a CF of 0.75

Rule cf9:

> **If** palpitations are present
> **then** HD_confirmation is true with a CF of 0.4

Assume that the following information is in the working memory before these rules are tried:

CF(cardiac symptoms present) = 0.6

CF(type of chest pain is cardiac) = 0.8

CF(palpitations present) = 1.0

All the antecedents in both the rules have CFs greater than 0.2. Therefore, both rules are applicable for the conclusion that HD_confirmation is true for the patient.

Assume that cf1 will be tried first and then cf9. In rule cf1, the CFs of the antecedents are 0.6 and 0.8. The minimum of the two is 0.6 which becomes the CF of the set of antecedents. The CF associated with the rule is 0.75. Hence the contribution for HD_confirmation from the rule cf1 is $0.6 * 0.75$, that is, 0.45. Similarly, from rule cf9, the CF contribution will be $min(1.0) * 0.4 = 0.40$. The CF of the hypothesis, HD_confirmation, is 0.0 to start with. After firing rule cf1, the new CF will be $0.0 + 0.45 * (1 - 0.0) = 0.45$, using F3.1.

After firing rule cf9, the combined CF for HD_confirmation is equal to

$0.45 + 0.40 * (1 - 0.45) = 0.67$, again using F3.1.

5.5.4 Merits and Demerits of Certainty Factors

The major objection against the use of certainty factors in expert systems is that they have no sound theoretical basis. MYCIN certainty factors are based mostly on intuition. Most of the time the only justification that can be provided for this approach is that "it works". There have been some attempts at formalising the approach based on conditional probability [Buchanan and Shortliffe, 1984]. But the *ad hoc-ism* remains.

There is a noticeable problem when you have many sources contributing positive and negative CFs to a conclusion. If there are a large number of small, positive contributions and a few large negative contributions, the net contribution could be closeto zero; this indicates equal evidence for and against. That is, the number of contributions for and against are not well reflected in the approach. This effect is particularly noticeable near the positive and negative certainty point (that is, CF = 1.0 or −1.0).

Another conceptual problem is the interpretation of a CF of 0.0. It could arise from equal evidence for and against (which indicates conflict) or because nothing is known about the conclusion (which indicates ignorance). The CF approach is unable to distinguish between the two cases.

Recent attempts at understanding CFs have shown that the approach can be considered a special case of general probability theory, with some simplifications. These simplifications were introduced to circumvent the main problems in implementing the probability based approach. But it has been proven that the simplifications introduced require underlying assumptions of symptom independence, uniform *a priori* probability etc. Hence the CF approach also suffers from some of the drawbacks of the probability approach, regarding the validity of such assumptions.

Another observation that can be made about the CF approach is that minor variations in the CF of the data or CF attached to the rules do not seriously affect the final results. Since the experts or the end users are not used to the idea of using numerical figures to denote uncertainty, any figures obtained from them will only be an estimate with a reasonable range of tolerance. Hence this insensitivity to minor variations in CF is one of the major plus points of the CF approach.

The CF approach has been quite popular among expert system implementors. Several shells and applications have adapted the approach for their systems. The main attraction of this approach is the intuitive basis and the fact that it has been shown to work in a real system. As mentioned earlier, this approach can be easily used in a rule based reasoning system.

The CF approach does not have the problem of the same evidence contributing both for and against a hypothesis, which we encounter in probability based approaches (see Section 5.4.2). A positive CF indicates a net belief *for* a hypothesis whereas a negative CF indicates a net belief *against* the hypothesis. And unlike in the probability based approach the evidence against and the evidence for a hypothesis do not have to add up to 1.0.

Summary

- In an expert system application, uncertainty can arise from various sources: inaccuracy in measurements, inherent imprecision of the language in which knowledge is being communicated, incompleteness of information, multiple information sources etc.

- There are two classes of uncertainty handling methods: numeric and symbolic. The numeric model handles if-then rules of the form *If condition then conclusion with certainty X* and attaches uncertainty measures with known as well as deduced information.

- Conditional probability with Bayesian reasoning provides a formal way of handling uncertainty (e.g., PROSPECTOR). But the large amount of data required, the non-intuitiveness of a purely numeric approach and some semantic problems makes this approach unpopular for expert system applications.

- Certainty Factor (CF) based methods resolve some of these problems and are widely used in expert systems. CFs were first introduced in MYCIN. The general class of such methods can be characterised using three formulae, as shown in Figure 5.1.

- MYCIN's formulae (Figure 5.2) are based mostly on intuition and have many nice properties. In addition, MYCIN also has a thresholding mechanism which helps to reduce the search space by pruning away unlikely hypotheses.

Exercises

1. Assume that you are developing an expert system for troubleshooting a complex electronic system. What are the sources of uncertainty that can arise in this application?

2. Describe briefly how the Bayes' theorem can be used for uncertainty handling in Expert Systems?

3. In which of the following situations would statistical reasoning be more valuable than CF-based (inexact) reasoning? Why?

   (a) Playing poker

   (b) Speech recognition

   (c) Electronic circuit fault diagnosis

   (d) Predicting the chances that a particular act will be passed by Parliament

4. In a survey conducted in the city of Bayesbad, it was found that

   - 20% of the people had disease D.
   - 45% of the people complained of headache.
   - 15% of the people suffered from D as well as the headache.

   What is the probability that a person picked at random has disease D? What would your answer be if you knew he had a headache?

5. Is it possible to compute p(d | not s) if you are given p(d), p(s | d) and p(s)? If so, how? If not, why not? You should not make any further assumptions about the data.

6. Design another framework for certainty factors by creating a set of formulae different from those in given in the chapter. Compare your framework with the MYCIN framework using some examples.

7. Consider the set of rules

P1: **If** aircraft is hostile
then it is a potential threat (CF=0.2)

P2: **If** aircraft is approaching
and it has ignored warnings
then it is a potential threat (CF=0.5)

P3: **If** aircraft-type is military
then it is a potential threat (CF=0.25)

(a) In one run, the following CF values were observed for the various antecedents:

aircraft is hostile $\rightarrow$ 0.9
aircraft is approaching $\rightarrow$ -0.5
aircraft-type is military $\rightarrow$ 0.4

i) Do all the rules qualify to fire in this case? Why?
ii) Do you need the values for the remaining antecedents?
iii) What are the CF contributions from each of the three rules?
iv) What is the resulting CF for the hypothesis that the aircraft is a potential threat?

(b) In another run, *potential threat* was found to have a CF of 0.4. The CF values for some of the antecedents are given below:

aircraft is approaching $\rightarrow$ 1.0
aircraft-type is military $\rightarrow$ 0.5
it has ignored warning $\rightarrow$ 0.7

Are these values possible? If so, what is the CF for the antecedent *the aircraft is hostile* in rule P1?

6. Explanations and the Human Computer Interface

6.1 Introduction

When you are not well, you go to a doctor. The doctor would examine you, arrive at a diagnosis and prescribe a course of treatment. If you are worried, as many of us are, you may ask the doctor what is wrong with you. A good doctor would answer your questions, and perhaps even tell you what hypotheses he considered for your problem, and how he rejected some or all of them. In some cases, he may ask for some laboratory tests to be done to differentially diagnose your problem. In this case, he probably is considering alternative hypotheses; he is trying to obtain additional information about your condition, so that he can zero in on a diagnosis.

Compare this with a doctor who gives you a prescription, and refuses to answer any reasonable questions. It is hard to trust such doctors, who expect you to have unquestioning faith in them.

We would all agree that a doctor who explains what he suspects, and what he is doing to control the problem, is better than a doctor who avoids explanations. The former is more credible.

Now, if it were a computer program doing an analysis and coming to a conclusion, it is all the more important that there be a high degree of credibility associated with the program. This is one reason why generating explanations is an important issue in expert systems.

The conclusion that an expert system reaches is supported by a chain of reasoning; this chain of reasoning can be used to justify the conclusion. The system should provide a simple way of communicating this to the user. For any program to communicate well with a user, it must have a good human computer interface — expert systems are no exceptions. The interface must be tailored to the task at hand, and must possess a variety of other attributes.

In this chapter, we will study the utility of explanations, desirable characteristics of explanation sub-systems in expert systems, and common mechanisms of providing explanations. Annotated examples of explanations provided by Vidwan (see Appendix A for information on Vidwan) will be used to illustrate the concepts being discussed.

We will then examine the characteristics of a good human computer interface of interactive programs in general, and of expert systems in particular.

6.2 The Utility of Explanations

Explanations can also be useful as a tool to help the Knowledge Engineer (KE) refine the Knowledge Base (KB) of an expert system. When a KB is being developed for an application domain, the KE would use test cases to verify the validity of the knowledge. If explanations are available, the KE can verify the accuracy of the KB, by tracing the knowledge used by the system to come to a conclusion. Chapter 9 provides more information about the validation and refinement of rule bases.

Some systems allow the user to have control over the reasoning process employed. These systems are usually very difficult to verify or correct, unless there are traces of the path of reasoning followed. Explanation traces can be used to understand the reasoning process followed, and to change it, if necessary.

Explanations can also act as a tool for debugging. Typically, the view of the rule base that is available to the KE is the view offered by the explanation sub-system. Usually, in any expert system, the KE can *step* through the execution of test cases. The KE will then have a better picture of the rules that are being used, rules which should have been considered but are not being considered etc. Sometimes (as we will see later in this chapter), it is useful to see why certain conclusions are not being reached. Explanations can provide some information about this as well. Using the information provided by explanations, the KE can then decide which rules need to be changed or modified to get a system to work correctly, or to enhance performance.

In systems where uncertainty reasoning mechanisms are used, explanations can provide insight into assigning proper values to certainty factors associated with rules.

As we had mentioned earlier, a system which provides reasonable explanations of its behaviour is far more credible than one which has no such facility.

We shall see later that the explanation facilities provided by present day expert systems are fairly simple-minded. Nevertheless, they satisfy many of the potential uses of explanations given in this section.

6.3 Explanations – Why and How

Expert systems today provide primarily two types of explanations[1]. These have been discussed in the literature as *Why* and *How* explanations. The following sections describe these types of explanations using an example expert system called Hruday. Hruday is one of the case studies in this book, and is described in Chapter 11.

Hruday is a Heart Disease Advisor, designed to provide advice to a non-specialist user. The system is not designed to play the role of a cardiologist who provides a prescription or specifies a course of treatment. Rather, it behaves as a knowledgeable friend who asks a person a few relevant questions and gives him relevant and helpful comments. Hruday has been developed using the expert system shell Vidwan [NCST, 1993].

[1]See [Moore and Swartout, 1988] for an excellent survey on explanations.

6.3.1 Asking Why

When an expert system requests some data from the user, he may ask the question: "*Why* is this data required?". One way to answer the question is to show the user the general direction that the expert system is headed towards. It is useful to tell the user what the current goal is. The system could say something like: "I am currently trying to establish (goal) X. In order to prove X, I need to know if (sub-goals) X1, X2 ... are valid. To establish the truth of X2, I need information about parameter X21..."

Consider X as the root of a tree, and paths from X to X1, X2 etc., as well as paths from X2 to X21 etc. as branches of the tree. Parameters X1, X2 etc. are represented as nodes in this tree. We call this tree of parameters the *goal-tree*. This tree captures the dependencies among parameters. Finding values of parameters represented in this tree involves tree traversal, and what we term *tree search*. We say we are going *down* the goal-tree when we move from goal to sub-goal to sub-sub-goal and so on. Similarly, we can say we are going *up* the goal-tree, when we go from the more specific to the more general, from a sub-goal to the goal.

Consider the rule with rule-identification number *cf11* in the expert system Hruday:

```
rule (cf11, HD_confirmation, true, 0.20) :-
        is(complaint, swelling_of_feet),
        gt(age, 15.0),
        lt(age, 55.0),
        is(sex, female),
        isnot(pregnant, yes).
```

The English paraphrase of this rule says:

Rule: cf11
If complaint is swelling of feet
and age is in the range 15 to 55
and sex is female
and patient is not pregnant
then conclude HD_confirmation with CF = 0.2

While evaluating the antecedent of the rule, the system might ask the user "Is the patient pregnant?". The user may wonder why the system wants to know if the patient is pregnant. The system may then explain that it is needed to confirm the possibility of heart disease, in the context of the evidence available so far.

The medical knowledge used in defining this rule is the following: Patients with certain types of heart diseases may exhibit, as one of their symptoms, a swelling of the feet. However, this symptom is also common in pregnancy. So the question "Is the patient pregnant?" is used to distinguish patients who have swelling of feet due to cardiac problems, from those who exhibit this symptom because of pregnancy.

6.3.2 Asking How

When an expert system reaches some conclusion, the user may want to ask: "*How* did the system reach this conclusion?". This might involve explaining the reasoning involved in understanding and diagnosing the problem, and in formulating and executing problem solving strategies. The system may have to explain how intermediate conclusions are arrived at, and how parameter values are acquired or inferred. This essentially involves tracing the steps that the system took to reach the conclusion.

In particular, it will mean moving from specific conclusions through rules that led to these conclusions, to the data supplied by the user. During this process, the user will be going down the goal-tree.

A good explanation system has to provide reasonable answers to the user's *Why* and *How* questions.

Note that, by its very nature, the *Why* feature is useful primarily in backward chaining systems. This cannot be provided in any simple manner for forward chaining systems.

6.4 Mechanisms for Providing Explanations

There are two very commonly used mechanisms for providing *Why* and *How* explanations in expert systems: using canned text or templates, and using *natural language* translations of the rules/code used in solving the problem.

6.4.1 Explanations using Canned Text or Templates

In the canned text approach, the designer of the expert system anticipates all the questions that may be asked. He attaches a ready-made (*canned*) explanation corresponding to each such question. Thus, for each parameter that requires a value, there is a natural language text segment that explains why that value is required at that particular juncture.

In the example given in Section 6.3.1, there can be a segment attached to the parameter `pregnant` saying: "Pregnant women often have swelling in their legs. We have to distinguish between pregnancy and heart disease as the cause for the swelling in the patient's legs. This is why we need to know if the patient is pregnant."

The canned text may sometimes contain slots for the values of parameters. These values are obtained from the user during the interaction, or computed from other rules. Canned text containing slots that have to be filled in at run time are called *templates*. For example, in Vidwan, we may have messages of the form: "The patient's diastolic pressure, ^DP^, is greater than the acceptable level....". Here, ^DP^ is a place holder where the system will insert the current value of the parameter DP, at the time the template is used.

Advantages and Disadvantages of Canned Text

Explanations involving canned text and/or templates are easy to prepare, and are easily used where they are required. They can be tailored to specific contexts, and may be controlled fully, unlike code translations which we will discuss next. However, there are several drawbacks to this method of providing explanations.

First, the designer of the expert system has to anticipate all possible questions. While it may be slightly simpler to answer *Why* questions in this format, it will be quite difficult to anticipate and answer all possible *How* questions.

When an expert system is updated, all the relevant explanation templates or explanation text have to be updated as well. If these updates are not properly carried out, the system may become inconsistent, with no proper correspondence between the code and the text of the explanations.

At a different level, this method of explanation is unsatisfactory because there is no clear link between the explanations and the behaviour that is being explained. The explanations themselves are not linked to one another. A complex action does not have a series of connected explanations; what we get is a series of possibly disjoint segments of explanations. Each segment may explain some portion of the reasoning process, but together they may not account for the entire path of reasoning.

Thus this method is useful for small expert systems, where the number of possible *How* and *Why* questions are limited and predictable.

6.4.2 Explanations using Code Translations

In the code translation approach, the system keeps track of the code (or rules) used to arrive at a particular solution point. If the user requests an explanation at any point in the interaction, the system converts the code used to a form closer to English (or some other natural language). In rule based systems, the system would keep track of the rules examined in order to arrive at a conclusion. If the user asked a *How* question, the system would trace the rules in the reverse order of the rule firing, to show how that conclusion was reached.

To answer a *Why* question, the system would use the goal attribute of the current rule as the current (sub-)goal, and give an explanation such as:

The value of "complaint" is required to establish HD_confirmation. This is because, according to rule cf11, to establish HD_confirmation, the following have to be established:

complaint is swelling of feet
and age is in the range 15 to 55
and sex is female
and patient is not pregnant

Examples of explanations using rule translations are given in Figures 6.1, 6.2 and 6.3. These are actual examples, from Hruday, that have been annotated to be informative.

Figure 6.1 shows the use of *Why* questions in Vidwan to find out the reason a particular parameter (say P) is being prompted for. Typically, the value of P will be required to find the value of P1, and the value of P1 will be required to find out P2 etc., till Pn, the goal attribute. The value of parameters P1, P2 etc. thus depend on the value of P. Using *Why* questions recursively, the user can trace the parameters which depend on the value of P. These explanations provide one particular view of the rule base, which may be of use to a KE.

Figure 6.2 shows *How* questions being used to find out how Hruday reached a particular diagnosis. *How* questions may be used recursively, to find out the relationship between the diagnosis and the intermediate steps in the diagnosis, and how these are linked to the data supplied by the user.

Figure 6.3 shows a different use of *How* questions. The KE can use *How* questions to find out why a particular diagnosis was *not* reached. This is useful to trace diagnosis paths for correctness, and to close gaps in the reasoning chain.

Advantages and Disadvantages of Code Translations

There are several clear advantages of using code translations.

It is very easy to implement this scheme, particularly for rule based expert systems. Some very simple mechanisms give an aura of intelligence to the explanation system.

The explanation provided by the system changes with the code. That is, if the rule base is modified in any way, the explanations associated with the rule base also change automatically, since the explanations are generated anew every time.

The expert system designer does not have to foresee all possible questions. Many of them will be adequately handled by the general explanation mechanisms.

Thus, this method of providing explanations is useful in a variety of expert systems. The disadvantages are also fairly clear.

There is still no high level cognition of the nature of the explanation provided. The system simply does not *understand* the explanation that it provides. Deep questions may be incorrectly answered with irrelevant detail.

The answers provided are uniform, and are not adapted to the particular problem being solved, or the depth of domain knowledge that the user possesses.

The system has no real understanding of the nature of the rule base. For example, rule cf11 above may be interpreted to mean that `HD_confirmation` is relevant (only) to non-pregnant women between the ages of 15 and 55, who have swelling of feet. But, actually, this rule is only meant to rule out the possibility of the effects of pregnancy being mistaken for a cardiac condition. To establish pregnancy, we must first establish that the patient is indeed a female, and of the appropriate child-bearing age. Thus, conditions C2 and C3 are to ensure that only if the possibility exists will a patient be asked if she is pregnant.

Rule: cf11
If complaint is swelling of feet C1
and age is in the range 15 to 55 .... C2
and sex is female C3

This is an annotated excerpt from an interaction with Hruday, using Vidwan. The patient's responses are given in **bold** letters. Annotations are in *italics*, and all other text is from Hruday.
All copyrights and caveats that apply to Vidwan and Hruday apply to this excerpt as well.

Is your chest pain felt mainly during coughs?
>>**@**
Hruday *enters the explanation mode*

<<<Vidwan – Explanation Mode>>>
Option:: **why**
The user asks "Why do you want to find out if the chest pain occurs during coughs?"

I am trying to evaluate the attribute chest_pain_type
For that I am using the rule ch6
RULEID: ch6
If complaint is chest_pain
and cough_linked_cp is true
then conclude that
chest_pain_type = cardiac with confidence -0.19
This involves a straightforward lookup to the goal of the current rule, which is rule ch6.

Option:: **why**
Why do you want to find out about chest_pain_type?

I am trying to evaluate the attribute HD_confirmation
For that I am using the rule cf1
RULEID: cf1
If card_symp_present is true
and chest_pain_type is cardiac
then conclude that
HD_confirmation = true with confidence 0.75

Now we can go recursively up the goal tree, in this manner, till we get to the top-level goal
Option:: **why**
Why do you want to find out about HD_confirmation?
I am trying to evaluate the attribute suspected_disease
For that I am using the rule sd4

Figure 6.1a: *Why* Questions in Hruday, using Vidwan.
An Annotated Example of the Use of *Why*.

RULEID: sd4
If card_symp_present is true
and HD_confirmation is true
and IH_risk isnot true
and val_surgery is true
then conclude that
suspected_disease = RHD with confidence 0.80

Option:: **why**
Why do you want to find out about suspected_disease?

I am trying to evaluate the attribute disease
For that I am using the rule dis2
RULEID: dis2
If suspected_disease is IHD
and IH_risk isnot true
and age gt 40.0
and bypass_surg is true
then conclude that
disease = IHD with confidence 1.00

Option:: **why**
Why do you want to find out about disease?

I am trying to evaluate the attribute adv
For that I am using the rule a3
RULEID: a3
If HD_confirmation is true
and bmi gt 30.0
then conclude that
adv = diet with confidence 0.90

Option:: **why**
Why do you want to find out about adv?

What do you mean why? That is my goal attribute
The system has reached the top of the goal tree. "adv" was given as the goal attribute in this consultation, and the system now indicates the fact that it cannot go up the goal-tree any further.

Option::**back**
Go back to the consultation mode.
Is your chest pain felt mainly during coughs?
>>-0.8
[...]

Figure 6.1b: *Why* Questions in Hruday, using Vidwan.
An Annotated Example of the Use of *Why*. (Contd.)

This is an annotated excerpt from an interaction with Hruday, using Vidwan as the shell. The patient's responses are shown in **bold** letters. Annotations are in *italics*, and all other text is from Hruday.
All copyrights and caveats that apply to Vidwan and Hruday apply to this excerpt as well.

First, we take a look at Hruday*'s diagnosis of a particular patient's condition. Since we are concentrating only on the explanation now, we will not discuss the structure of the diagnosis at this point.*

Advice to patient
You show evidence of ischaemic heart disease. Avoid all unusual physical ef fort.
You need a cardiologist's attention. (0.24)
Avoid mental strain and tension where possible. (0.22)
Now we study the diagnosis. There are a number of options available here. We will look at only a few of them.

Enter your option: **how**
Enter the attribute name: **adv**
The user asks "How did you establish the values of adv?"

The system prints out the values and certainty factors contributed by each relevant rule that was applicable in this case. For example:
The applicable rules and their contributions are as follows:

Value IHD_attn from rule: a5 with CF 0.27
RULEID: a5
If disease is IHD
then conclude that
adv = IHD_attn with confidence 1.00
[...]

At this stage, the system also prints out the rules which could have fired but did not fire in this patient's case. In each case, the antecedent that failed is listed. This part of the system's output has been omitted in this excerpt.

Figure 6.2a: *How* Questions in Hruday, using Vidwan.
An Annotated Example of the Use of *How*.

Enter your option: **how**
Enter the attribute name: **disease**
Again, we are going down the goal-tree. From the previous question, it was clear that adv got one of its values based on a value of disease. Now we find out how disease got its value.

The applicable rules and their contributions are as follows:
Value IHD from rule: dis1 with CF 0.27
RULEID: dis1
If suspected_disease is IHD
and HD_confirmation is true
then conclude that
disease = IHD with confidence 1.00
Other applicable rules are also shown at this stage, but have not been reproduced here.

Enter your option: **how**
Enter the attribute name: **suspected_disease**
Now go down to other levels of the path to the diagnosis.

The applicable rules and their contributions are as follows
Value IHD from rule: sd1 with CF 0.27
RULEID: sd1
If card_symp_present is true
and IH_risk is true
then conclude that
suspected_disease = IHD with confidence 0.60
[...]

Enter your option: **how**
Enter the attribute name: **card_symp_present**
The applicable rules and their contributions are as follows
We have 4 sources of evidence, each contributing to the overall certainty. Only one is shown here.

Value True from rule: cs3 with CF 0.18
RULEID: cs3
If complaint is swelling_of_feet
then conclude that
card_symp_present = true with confidence 0.30
[...]

Figure 6.2b: *How* Questions in Hruday, using Vidwan.
An Annotated Example of the Use of *How*.(Contd.)

Enter your option: **how**
Enter the attribute name: **IH_risk**
The applicable rules and their contributions are as follows
We have 5 applicable rules for IH_risk.

Value True from rule: ihr2 with CF 0.15
RULEID: ihr2
If sedentary is yes
then conclude that
IH_risk = true with confidence 0.15

Value True from rule: ihr7 with CF 0.10
RULEID: ihr7
If sex is male
then conclude that
IH_risk = true with confidence 0.10
[...]

Three more rules are shown.

Enter your option: **how**
Enter the attribute name: **sex**
Value: male from User
The system says that it got this value from the user during the interaction.

Enter your option: **how**
Enter the attribute name:**age**
32.0 from user
[...]

Figure 6.2c: *How* Questions in Hruday, using Vidwan.
An Annotated Example of the Use of *How*.(Contd.)

This is an annotated excerpt from an interaction with Hruday, using Vidwan as the shell. The patient's responses are given in **bold** letters. Annotations are in *italics*, and all other text is from Hruday.
All copyrights and caveats that apply to Vidwan and Hruday apply to this excerpt as well.

In this case, we simulated a patient with no cardiac disease. However, we wish to find out why the patient had no "applicable top level" rules. So, we are actually doing a sort of **How-Not** *query – How did* Hruday **not** *diagnose any heart disease?*
As in Figure 6.2, first we have the diagnosis.

Advice to patient

No applicable top level rules

Enter your option: **how**
Enter the attribute name: **adv**
That attribute has no value at the moment

The goal attribute has no value. The following sequence tells the user why rules which could have fired did not fire in this patient's case. In each case, the antecedent that did not fire is listed. This helps the KE debug the system, and helps the user understand why the rule did not fire.

The applicable rules and their contributions are as follows

Rule a3 has failed to fire
The first antecedent that failed is: 1
RULEID: a3
If HD_confirmation is true
and bmi gt 30.0
then conclude that
adv = diet with confidence 0.90

Rule a301 has failed to fire
The first antecedent that failed is: 1
RULEID: a301
If HD_confirmation is true
and bmi gt 24.0
and bmi le 30.0
then conclude that
adv = diet with confidence 0.82
[...]

Many other rules which did not fire were also listed.

Figure 6.3a: *How-Not* Questions in Hruday, using Vidwan.
An Annotated Example of the Use of *How*.

Enter your option: **how**
Enter the attribute name: **disease**
Now the user goes to another level, to find out why antecedent 1 in Rule a301 failed.

That attribute has no value at the moment
The applicable rules and their contributions are as follows

Rule dis1 has failed to fire
The first antecedent that failed is: 1
RULEID: dis1
If suspected_disease is IHD
and HD_confirmation is true
then conclude that
disease = IHD with confidence 1.00

Rule dis2 has failed to fire
The first antecedent that failed is: 1
RULEID: dis2
If suspected_disease is IHD
and IH_risk isnot true
and age gt 40.0
and bypass_surg is true
then conclude that
disease = IHD with confidence 1.00
[...]

More rules were listed, that did not fire. Now we look at an attribute that does have a value.

Enter your option: **how**
Enter the attribute name: **suspected_disease**
The applicable rules and their contributions are as follows

Now we locate some rules that fired, and some that did not, for the attribute: suspected_disease. In each case where it did not fire, the system tells you which antecedent failed.

Value IHD from rule: sd1 with CF 0.12
RULEID: sd1
If card_symp_present is true
and IH_risk is true
then conclude that
suspected_disease = IHD with confidence 0.60

Figure 6.3b: *How-Not* Questions in Hruday, using Vidwan.
An Annotated Example of the Use of *How*.(Contd.)

Value RHD from rule: sd3 with CF 0.04
RULEID: sd3
If card_symp_present is true
and age gt 10.0
and age le 40.0
then conclude that
suspected_disease = RHD with confidence 0.18
[...]

More rules follow, some which failed and some which succeeded. Even though some rules fired, note that the values of suspected_disease – IHD/RHD/CHD – all have CF values much below the threshold of 0.2. They are therefore ignored as insignificant.

Enter your option: **how**
Enter the attribute name:**card_symp_present**
The applicable rules and their contributions are as follows

Rule cs3 has failed to fire
The first antecedent that failed is: 1
RULEID: cs3
If complaint is swelling_of_feet
then conclude that
card_symp_present = true with confidence 0.30
[...]

skipping a list of rules which failed

Value True from rule: cs8 with CF 0.20
RULEID: cs8
If complaint is chest_pain
then conclude that
card_symp_present = true with confidence 0.40
This tells us that attribute card_symp_present (cardiac symptom present) is true because of the complaint of chest pain. However, there are no other factors which conclusively point to heart trouble, and therefore there is no conclusive diagnosis for the patient.

Enter your option: **quit**
We found out why we did not get a conclusive answer, so we quit!

Note that, in general, the rule base must be so designed that the message "No applicable top level rules" is never displayed to the user. The KE must ensure that some meaningful message is given for all possible situations. In this specific example, we could have some message such as "I do not have a conclusive diagnosis for the patient. Please consult a doctor" displayed.

Figure 6.3c: *How-Not* Questions in Hruday, using Vidwan.
An Annotated Example of the Use of *How*.(Contd.)

and patient is not pregnant C4
then conclude HD_confirmation with CF = 0.2

However, the code translation system may not understand this. The conditions C2 and C3 would not be seen as filtering conditions, but as necessary preconditions to establish the goal. Thus, in all likelihood, the system would provide explanations which are unsatisfactory.

There is another reason why code translations may not be useful. The code is written, in general, to optimise program performance. It is not usually designed to be meaningful by itself, even if rendered in English. For example, consider the response to a *Why* question involving condition C2 of rule `cf11`. It would not be very useful just to say that the query is to find out if the patient is between 15 and 55 years of age. A better response would consider this age check as part of the larger goal of finding out if the patient is pregnant. The response could then be that the system is trying to establish if the patient is pregnant. The average explanation system is not likely to provide this explanation. Note that a good explanation need not use the code corresponding to all the conditions.

The reader may suggest that rule cf11 could be split up into one rule which checks if the patient is pregnant, and another, which uses this information to judge if heart disease may be confirmed. The pregnancy checking rule can have the age and gender checks to avoid asking irrelevant questions. This will also ensure that *Why* questions about pregnancy and HD_confirmation will be answered with better explanations.

Ideally, explanations should be independent of the structuring of the rule base, and should depend only on the domain structure. However, the rule base structure that is adopted by a KE depends on the tool(s) that he uses to create the expert system. Thus, the structure used is a particular encoding of domain knowledge, and may not accurately reflect the domain structure. However, the closer the rule base structure is to the domain structure, the better the explanations are likely to be.

Explanations using code translations are unfortunately too closely linked to the code, and are therefore not very useful in providing convincing, understandable explanations.

To sum up, explanations using code translations are better than canned text systems. While they are unsatisfactory in various ways, they are adequate for many purposes and are widely used in expert systems today.

6.5 Research Areas in Explanation Systems

Moore and Swartout argue that experts generate explanations [Moore and Swartout, 1988] – they do not use canned text, or translate their chain of thoughts into English. It is also possible that they come up with a new explanation as a consequence of the problem solving process. Thus, Moore and Swartout claim that we need to look at explanations as a separate and important component of expert systems. This line of research is promising, though it is not clear if such explanation systems are feasible in the near future.

MYCIN [Shortliffe, 1976] provided simple rule-tracing aids for the KE. As an improvement to MYCIN, EMYCIN [van Melle et al, 1984] provided facilities such as Explain, Test and Review, which were meant to help the KE in debugging the rule base. The use of meta-rules in MYCIN and EMYCIN made it possible to think of meta-level reasoning about the strategies used in problem solving.

The knowledge in MYCIN was restructured in a later system called NEOMYCIN [Clancey and Letsinger, 1981] to make it suitable for tutoring purposes. NEOMYCIN attempted to provide explanations at the level of strategies used in problem-solving. The basic knowledge organization in NEOMYCIN was oriented towards this. The medical knowledge and the problem-solving strategies were separated. There was an explicit representation of diseases, in a taxonomy. The rules linking data to hypotheses were separated from this. MYCIN's backward chaining was replaced by the execution of meta-rules in NEOMYCIN.

Centaur [Aikins, 1983], a reimplementation of the PUFF expert system also attempted to provide for sophisticated explanations. We saw in Section 4.4.4 that Centaur worked by using the initial data to trigger prototype instantiation, and selecting one prototype as a candidate prototype for further exploration. Using known facts to fill the prototype, Centaur compared expected values of parameters with actual values, and used the discrepancy to refine the diagnosis.

Centaur operated in stages; at each stage Centaur made the context clear, (showing the prototype being considered, for example) and provided a lot of information about its findings. This information included listings of findings that were consistent, inconsistent or not accounted for by each prototype considered. Such detailed information helps the KE in debugging the rule base.

All these approaches are interesting, and have resulted in domain specific solutions to the problem of proving better explanations.

6.6 Desirable Characteristics of Explanation Systems

We have seen that the explanation facilities provided by current day expert systems are rather limited. Explanation systems can answer only limited types of queries. They are often inflexible, in that explanations can be presented only in one way. They are not adapted to the knowledge levels or needs of different users. They do not provide different levels of explanations, and are often inextensible.

However, there is considerable research interest in this area. The next generation of expert systems would hopefully be capable of generating better explanations.

What characteristics would we look forward to, in a system that would provide better explanations?

We would like explanations to be *flexible* and *adaptable*. Explanations should be tailored to particular situations, and particular problems. Explanations should be dynamically generated to provide accurate feedback in response to specific user queries.

Explanations should be tailored to the user's knowledge of the domain. An expert may not need a detailed explanation, whereas a novice may need all the detail possible, and would prefer a fine grained explanation. Therefore, it should be possible to get explanations at different levels of granularity.

It would be better still if the system were to adapt itself to the user's comprehension of the explanation provided. Thus, the system may start by providing a certain level of detail, and provide less or additional detail depending on whether the user has understood the explanation or not. This brings in issues of user modeling, efficacy of explanations, etc., which are still topics of research interest.

It should be possible for the explanation system to build upon explanations given earlier in a session. Thus, explanations could be made structured.

Explanation systems should also use all possible modes of interacting with the user to provide the best possible explanation in a particular situation. If graphs, charts or a video clip would explain a situation better, the system should use such aids. All this assumes that a good human computer interface exists. We shall examine such interfaces in the next section.

To sum up, present day explanation systems have a variety of limitations. Current research is aimed at developing systems that are capable of providing better explanations.

6.7 The Human Computer Interface

Human Computer Interfaces (HCIs), also known as user interfaces, are important in two different contexts in expert systems. During the creation of expert systems, the KE should be provided with a good interface (by the expert system shells). Later, during the actual use of the expert systems, the user should be provided with a good interface.

Many of our comments on human computer interfaces are applicable to a wide variety of interactive systems, and are not limited to expert systems. However, in an area such as expert systems, where systems solve critical problems, it is all the more important to achieve usability along different dimensions. Since the interface component is critical to any such system, it is important to have a good, user-friendly interface. Ideally, we would like an interface which:

- talks the same language as the user,
- uses the same units as the user,
- is adaptive,
- uses multi-modal communication, and
- allows for degrees of flexibility.

In addition, this interface should satisfy three important requirements [Nievergelt and Weydert, 1980]. Good user interfaces ought to tell the user, at any point of time:

- **Where he is**: What part (or stage) of the program is the user currently at?
- **How he came there**: How did the user come to this point? Is there a history of user commands?
- **What he can do at this point**: What are the options open to the user at the current point? Is there a menu which tells the user what actions are possible?

These issues are more related to facilities provided by expert system shells used to create expert systems, rather than to individual expert systems.

Readers interested in finding more about interface design may look at [Shneiderman, 1987] or [Dumas, 1988]. Shneiderman presents a survey of the human factors that need to be considered while designing interactive computer systems. Both Shneiderman's book, and the book by Dumas, provide sets of guidelines for designers of interactive systems. While a lot of these guidelines may seem to be homilies or well-understood truths, adherence to such guidelines may decide whether a package is a success or a failure.

More details about the attributes of a good interface are given in the subsections that follow.

6.7.1 Talking the Same Language as the User

In an interactive system, it is important to keep the nature of systems' queries at approximately the same level of discourse as the user's responses. If this is not done, there is every chance of the user misunderstanding the question, and providing incorrect, incomplete or irrelevant information.

If an expert system is to be built with doctors in mind, it is valid to ask: "Does the patient suffer from dyspnoea?". This question may not make sense to lay people. If the expert system is to be built with patients in mind, the same query could be rephrased as: "Are you breathless even after mild exercise?"

Adequate care should be taken in framing questions for any system. The wording of the prompt may misguide the user to provide an answer in a particular fashion, which may be different from what was intended.

For example, a query about a person's alcoholic intake (in a system such as Hruday, meant to be used by lay people) may be worded in various ways:

A1. Is the patient an alcoholic?

A2. Does the patient consume large amounts of alcohol?

A3. Does the patient consume more than 3 units of alcohol every day, on the average? One unit of alcohol may be taken as one peg of strong liquor or one glass (200ml) of beer.

Here, version A1 is too vague, and therefore confusing. Besides, the term *alcoholic* is loaded with medical and moral connotations, and people may hesitate to use the term to describe themselves.

Version A2 is better, but it is still not clear what *large* means. Is it one bottle of liquor a day? Or less?

Version A3 defines a specific threshold, and so is easier to answer. Thus, in framing such questions, it is perhaps better to be clear than attempt to be very brief.

6.7.2 Using the Same Units as the User

Interactive systems often need large amount of inputs from the user. Much of this information is related to numbers, such as “patient’s height”, “person’s weight”, etc. Care should be taken in framing questions that request such information.

Most parts of the world use the metric system. However, a few countries, such as the USA, still use the so-called British units. If you ask for the room temperature in the USA, you would get a figure in degrees Farenheit, while it is more likely to be in degrees Celsius in India, France etc.

People tend to use the units that they are accustomed to using. This could cause confusion. Hence a question to the user about a measurement should ideally include the units that you would like the answer to be given in.

Thus, you could say: “What is the weight of the patient (in Kgs)?” or “What is the cost of the filter (in Rupees)?”.

The unit used must be a unit the user is accustomed to, in the domain that it is used in. Thus you must avoid asking for distance between cities in *inches,* or cost in *Dutch guilders*, where *kilometres* and *Rupees* may make more sense. But it would be normal to ask for oil tanker size in *thousands of metric tons* or wavelength in *Angstroms*.

6.7.3 Being Adaptive

Interfaces have to adapt themselves to the user’s skill and knowledge level. This implies that a sophisticated model of the user must be maintained, which keeps track of the user’s generic skills, domain-specific skills and familiarity with the system.

At a very simple level, this may be reflected in the degree of menu selections versus command line inputs that are accepted from the user. Command line inputs are usually expected from experienced users while menus are preferred by novices.

At other levels, programs may offer different prompts and help levels, depending on the user.

A system which insists on being equally verbose or equally brusque with all users will be perceived as an unfriendly system.

6.7.4 Using Multi-modal Communication

Interactive input to programs can be in the form of text, a selection from a menu, or a command given in a restricted (command) language or pseudo-natural language conforming to some prespecified templates. It can also involve pointing with devices such as a mouse, or using iconic interfaces such as those used in the Apple Macintosh computers, PCs running MS-Windows etc.

The output from interactive programs can take various forms: text, near natural language, pictures, diagrams, charts, graphs, animation, video-clips, etc.

Ideally, a good interface will provide for the use of multi-modal input and output. For example, a system may accept input with a mouse and menu selections, and provide graphic output using animation to illustrate a maintenance procedure.

6.7.5 Allowing Degrees of Flexibility

A good interface will allow the user flexibility in configuring the interface, typically by allowing the user to switch on (or switch off) most features of the interface. Small items such as the volume of the warning bell may affect the perceived friendliness of a user interface. For example, the facility provided by some systems to turn the logging of interaction either on or off is a small but useful step in enhancing the user interface.

6.8 New Directions in Human Computer Interfaces

Computer scientists, cognitive scientists, psychologists, etc. today understand more about how we perceive and use interfaces. Advances in hardware technology are helping to create workstations with multiple modes of input and output. Advances in multimedia systems have made interaction come to life in a variety of ways. Research into user modelling may make it possible for interfaces to become more flexible, and adapt themselves to particular users' needs. Emerging standards in windowing systems may make programs more portable, since the interfaces themselves will be portable across a range of platforms.

Thus, while it may take some time, better interfaces are on the way!

Summary

- The capability to provide explanations is an essential feature of expert systems.

- Explanations are useful to understand the reasoning process used in an expert system. They are also useful in debugging, and in deciding on certainty factor assignments. A system which provides explanations is also more credible.

- Present-day systems provide very simple explanation features, usually *How* and *Why* features.

- The *Why* feature allows the user to ask why the system requires a certain piece of data.

- The *How* feature allows the system to trace its reasoning process, displaying intermediate steps and values of parameters at various points in the reasoning process.

- Explanations are provided with canned text (using templates), or by paraphrasing code to produce near-real English (or some other natural language). There are advantages and disadvantages to both these approaches.

- There are several features desirable in an explanation system such as adjusting to the user's level of knowledge, adapting to the granularity of explanation sought etc. Many of these are still open research issues.

- Human computer interface (HCI) issues are important during the development of expert systems, as well as at the time of actual use of these systems.

- Human computer interfaces need to talk the same language as the user, be adaptive to the user's requirements, use multi-modal communication methods, and be flexible.

- There is considerable research in HCI, aimed at making computer programs in general, and expert systems in particular, easier to use.

Exercises

1. Consider the Hruday rule base (see Appendix B). Identify at least three prompts to the user which are ambiguous or misleading. Explain why they are misleading. Suggest alternate wordings for these prompts.

2. Identify five points in the Hruday rule base where you feel the user may ask for *How/Why* explanations. Detail the canned explanations that you would like to provide the user at these points. If possible, describe unusual or rare circumstances where your canned explanations will not suffice, or will not adequately explain the diagnosis or the need for some item(s) of data.

3. List five units that are not normally used in everyday dialogues, but which are specific to particular professions. For example, a chemical engineer may think about temperature in the Kelvin scale. Mention the professions which would use these unusual units.

4. Consider the following rule base written using the expert system shell Vidwan [NCST, 1993] (see Appendix A for further details about Vidwan).

```
%  SHOOTP  -- To shoot or not to shoot
%  Copyright NCST, 1988-1992
%  Released for educational use

% This is the Vidwan rule base for an imaginary air
% defence system, meant for use on a ship. The expert
% system uses information such as:
%       * does the aircraft look hostile?
%       * does it belong to allies?
%       * is it approaching or receding?
%       * is it flying within allowed limits?
%       * what type of electromagnetic emission
%         is seen from the aircraft?
%        (mode1 indicates that it is a civilian aircraft;
%         mode2 indicates that it is a military aircraft).
%       * is the ship under attack from any source?
%
% The system has to help decide whether the commander of
% the ship should shoot the intruder aircraft, or warn
% the aircraft. The system evaluates the threat potential
% using multiple pieces of evidence, and uses that to
% determine its course of action.


% type definitions
type(action, m).
type(type1, p).
type(emission_type, p).
type(hostile, b).
type(potential_threat, b).
type(region_violation, b).
type(under_attack, b).
type(allies, b).

% goal
goal(action).

% menus
menu(type1, [approaching, receding]).
menu(emission_type, [mode1, mode2]).

% templates
template(hostile,
  "Is the aircraft hostile?").
template(type1,
  "Is the aircraft approaching or receding?").
template(region_violation,
  "Is the aircraft flying within allowed range?").
template(emission_type, "What is the type of emission
  received from the aircraft?").
```

```
template(under_attack,
   "Are you under attack from any source now?").
template(allies,
   "Does the aircraft belong to our allies?").

% notes
note(act1, "The aircraft may be SHOT DOWN").
note(act2, "The aircraft, though civilian, is flying out
   of range, issue a warning").
note(act3, "The aircraft seems to be a potential threat,
   issue a warning").
```

```
% rules
rule(pot1, potential_threat, true, 0.25) :-
        is(hostile, true).
rule(pot2, potential_threat, true, 0.25) :-
        is(type1, approaching).
rule(pot3, potential_threat, true, 0.25) :-
        is(type2, military).
rule(pot4, potential_threat, true, 0.15) :-
        is(region_violation, true).
rule(iff1, type2, military, 0.9) :-
        is(emission_type, mode2).
rule(iff2, type2, civilian, 0.9) :-
        is(emission_type, mode1).
rule(act1, action, shoot, 0.9) :-
        isnot(allies, true),
        is(under_attack, true),
        is(potential_threat, true).
rule(act2, action, warn, 0.4) :-
        isnot(allies, true),
        is(type2, civilian),
        is(region_violation, true).
rule(act3, action, warn, 0.5) :-
        isnot(allies, true),
        is(potential_threat, true),
        isnot(under_attack, true).
rule(act4, action, no_shoot, 0.8) :-
        is(allies, true).
```

(a) Mentally execute the rule base in an imaginary scenario in which a civilian aircraft of a hostile nation is shot down.

(b) Provide an explanation (by reasoning using the rule base) as to how such an action could have been taken.

(c) Try running the same rule base on Vidwan, if you have the software. When the system provides a conclusion, ask for an explanation. Compare your explanation with Vidwan's explanation for the same case.

How close is your explanation to Vidwan's explanation? Can you account for the differences, if any?

5. If we had an inference engine which was different from Vidwan, would we need a vastly different system for generating explanations? Can you think of some new method by which explanations can be generated, for a specific new inference engine? You have to specify enough about the new inference method to justify the new method of explanation.

7. Rule Based Systems in Use

Expert systems are no longer merely of academic interest. They have moved into commercial applications in a very big way. Expert systems have been one of the major commercial successes in the area of Artificial Intelligence. The market for expert systems was estimated at 900 million US dollars in 1990. Many industries have started using/exploring expert systems as a way of improving the speed and consistency of jobs, which have defied attempts at automation using conventional programs. Examples of companies using expert systems are General Motors, Ford, American Express, Du Pont, Boeing etc. [Feigenbaum et al, 1988] describes many expert systems used by such companies. The systems are discussed outlining the motivation for the system, the hurdles which had to be overcome in implementing the system, the approach followed and the economic as well as productivity payoffs realised.

Till early 1980, the only commercial expert system worth mentioning was XCON (earlier known as R1) from Digital Equipment Corporation (DEC). Certainly, there were many other systems being developed at other places. But few of them saw the light of day. Today DEC alone has more than 40 expert systems in use, saving more than 25 million US dollars a year.

In this chapter, we will look at a few example rule based expert systems. These should give the reader a feel for the applications, the complexity and effort involved, the approach followed and the knowledge representation methods used. Our stress in this book has been on rule based systems. Actually around 70 % of the expert system applications developed are rule based. Most expert system shells available commercially are rule based. Hence we will concentrate on rule based systems in this chapter. We will also concentrate on systems that have been put to real use, rather than systems of academic interest. There are a few seminal systems which do not pass this test, which nonetheless cannot be omitted in a book on expert systems. We will describe some of these as well.

7.1 DENDRAL

References: [Buchanan and Feigenbaum, 1978], [Carhart, 1979], [Barr and Feigenbaum, 1982]

DENDRAL is a fore-runner in the field of expert systems. Work on it was started in 1965 at Stanford University. The task was to determine the molecular structure of an unknown organic compound, based on molecular spectroscopic data. A Nobel Laureate in chemistry, Joshua Leiderberg, and the father of expert systems, Edward Feigenbaum, were the main architects of the system.

The steps involved in identifying an unknown organic compound start with obtaining information about the constituent atoms, etc., from various physical and chemical tests of the compound. In general, this is inadequate for determining the 3-D structure of the molecule, since one would have thousands of candidate structures satisfying such composition constraints. The compound is therefore subjected to spectrum analysis. During spectrum analysis, the compound is bombarded chemically, which causes some of the bonds in the molecular structure to be broken. The possible breakage points depend on the molecular structure of the compound. Using the results of such tests, a histogram showing mass distribution of the resulting components can be produced. This is a graph of the mass of resulting fragments against their relative abundance. For a given compound, this graph is nearly unique, but for "noise" and overlapping of various component fragments.

This histogram is analysed by experts to obtain clues about constituent fragments of the given compound. This would help experts to conclude about the presence or absence of certain groups, say Ketones, in the compound. These become a set of constraints to be satisfied by a valid structure for the compound.

Then the chemists would use brute-force enumeration to generate various possible candidate structures given the molecular formula of the compound (for example, $C_6H_{12}O_8$). Leiderberg had developed an algorithm to systematically generate all possible candidate structures. Each of the generated candidates is tested against the constraints obtained from the histogram. The number of such candidate structures was too large for this brute-force approach to be practical for many cases.

DENDRAL tried to capture the thumb-rules that experts use to quickly narrow down the possibilities in identifying the compound. The system consists of three parts named plan, generate and test, which we will describe one after the other.

Plan: This is a heuristic system which produces a set of constraints stating if certain groups such as Ketones should or should not be present in the compound. It uses the mass spectrometer-generated histogram to derive the constraints. The sample rule below illustrates the type of knowledge captured in this module.

> **If** there are two peaks in the spectrum at masses X1 and X2
> **and** X1 + X2 = M + 28
> **and** X1 − 28 and X2 − 28 are high
> **and** at least one of X1 and X2 is high
> **then** a Ketone group is present

M is the molecular weight of the compound, which is easily computed from the molecular formula.

Generate: This module, called CONGEN, systematically generates all complete structures of the given compound, using the constraints generated in the Plan module. By incorporating constraint checking in the generator itself, the number of candidate structures generated are reduced significantly. As soon as a partial structure fails to satisfy a constraint, it is discarded.

Therefore many structures, all of which would have anyway been discarded while testing for this constraint, are not generated.

Test: Given a complete structure, chemists can predict what kind of a histogram will be obtained for the structure, without actually using a mass spectrometer. This knowledge is used by the Test module to check if the histogram produced by the candidate structure (generated by the generator module) matches with the histogram obtained for the compound. The matches are ranked and presented to the user.

DENDRAL is currently maintained by Stanford University and chemists from all over the world access it through the computer communication networks.

7.2 PROSPECTOR

References: [Duda et al, 1979], [Barr and Feigenbaum, 1982]

PROSPECTOR was one of the earliest successful expert systems. Its domain was geology. It was used to provide consultation to geologists in early stages of investigating a site for ore-grade deposits. The system was developed at Stanford Research Institute (SRI) by Peter Hart, R Duda and others.

The system uses a semantic network (see Chapter 4) model for storing its knowledge. The nodes of the network represent assertions about the domain, such as *Quartz Monzonite is present*, *Favourable Level of Erosion*, etc. The arcs in the network can be viewed as inference rules. PROSPECTOR uses an uncertainty handling mechanism based on Bayesian probability. Each arc is labelled with two numbers - one representing how necessary the antecedent is for the consequent to be true and the other representing how sufficient the antecedent is for the consequent to be true. (We refer to the node at the tail of the arc as the antecedent and the node at the head to be the consequent). Please refer to Chapter 5 for more details of this aspect of PROSPECTOR.

When the probability of some node changes due to inputs given by the user, all nodes get their probabilities updated, based on the interconnections and the weight labels. The uncertainty figures visible to the user are mapped onto a scale from -5 to $+5$, with 0 indicating no evidence.

Early expert systems developed were meant for an academic environment and worried little about the user interface. PROSPECTOR was one of the first systems where user acceptance (from the point of view of ease of use) was looked at seriously. It had a restricted natural language interface which could accept sentences as input, enabling non-technical people to use the system. Also, it was one of the few systems which allowed the user to volunteer information, providing a mixed-initiative interaction mode. The system allowed the user to interrupt it any time in the run and volunteer information, change previous statements made or request evaluation of the current state.

The user would start interacting with the system by identifying some significant features of the site under consideration. For example:

There is Quartz Monzonite

Quartz Monzonite (5)

There may be biotite

Biotite(2)

The lines in *Italics* are the user's inputs and the lines in this font are system responses. The numbers indicate the certainty factor (in a scale from -5 to +5) inferred by the system from the user input.

The interaction is completely controlled by the user. Once the volunteering of information is completed, the system displays its current rating of the possibilities of various types of deposits in the area and allows the user to explicitly rule out some, if he so desires. The remaining are explored by the system in the order of likelihood. The user is prompted for information where required. When a hypothesis gets a certainty factor above 4.5, the system lets the user decide if this option needs to be explored further (since it is almost certain).

Finally, the system would list out the various hypotheses and the associated certainty factors.

In 1980, the system showed an example of its utility. Mount Tolman in eastern Washington was a partially explored site. PROSPECTOR was given data about this site. It predicted Molybdenum deposits in the area and the prediction was confirmed by actual drilling [Campbell et al, 1981].

Due to the academic setting in which it was developed, it was not put to real commercial use. SRI still uses the system for R&D purposes.

7.3 MYCIN

Reference: [Buchanan and Shortliffe, 1984]

MYCIN, which was developed at Stanford University, has been one of the most cited expert systems. A number of developments in expert systems were triggered by the advent of MYCIN. Examples are, the wide popularity of simple rule based representations, the concept of an expert system shell, use of certainty factors (see also Chapter 5), multiple uses of a knowledge base, etc. The main architects of the system were Bruce G Buchanan and Edward H Shortliffe.

MYCIN's domain was diagnosing bacterial infections of blood. It tracks down the organism causing the infection and prescribes a treatment. Generally in medicine one does not have sufficient information to make an absolute diagnosis. We do not know enough about human physiology and a doctor does not have access to all the required information most of the time. An ideal case for uncertainty reasoning! Probability based reasoning was found to be impractical – due to lack of sufficient data and more importantly, because experts do not relate to that kind of reasoning. Hence MYCIN architects introduced a method of handling uncertainty called the Certainty Factor method (see Chapter 5 for details). This method became very popular and a number of expert systems shells today provide some variation of this method.

MYCIN prompts the patient for standard information and then asks for specific pathology test results for narrowing down the hypotheses set. It has a fairly simple user model and provides *how* and *why* explanations (as described in Chapter 6). The knowledge representation is completely rule based. The rule syntax has a LISP flavour (LISP was the implementation language). A sample MYCIN rule is given below:

Premise:	($AND	(same cntxt gram gramneg)
		(same cntxt morph rod)
		(same cntxt air aerobic))
Action:	(conclude	cntxt class enterobacteriaceae tally 0.8)

A rough English translation of this rule is:

If	the stain of the organism is gramneg
and	the morphology of the organism is rod
and	the aerobicity of the organism is aerobic
then	there is strongly suggestive evidence (CF = 0.8)
	that the class of organism is enterobacteriaceae.

MYCIN was validated by comparing its diagnoses with the diagnoses made by medical students, ordinary doctors and expert doctors. The result was quite impressive, with MYCIN quite often exhibiting expert's level performance.

At Stanford, MYCIN marked the beginning of a variety of related activities. On the one hand, the TEIRESIAS system came out of the concern for knowledge acquisition and maintenance in these types of domains. TEIRESIAS provided many facilities for error checking of rules, interactive debugging of the MYCIN rule base and validation of new rules based on the rules already present in the system. We will examine this system in a little more detail in Chapter 10. More details about TEIRESIAS can be found in [Davis, 1979].

Since the expert systems approach advocates the separation of the domain knowledge and the inference mechanism, a natural question to ask is, "Can the knowledge base be used for tasks other than diagnosis?". For example, can the knowledge base be used for teaching students how to diagnose bacterial infections? This question was investigated by WJ Clancey, resulting first in a system called NEOMYCIN [Clancey and Letsinger, 1981] and later on in GUIDON [Clancey, 1987]. This work also led to some seminal results on the analysis of rule base structure in expert systems [Clancey, 1983; 1985]. It was noticed that the if-then rule structure in MYCIN is used to represent many different types of knowledge (for example, definitional, causal, associational and heuristic). For teaching purposes, it is important to distinguish between these different types of knowledge. Also it was found that the order of the rules in the knowledge base and the order of the conditions in a rule, explicitly represent some of the strategic knowledge (such as try this class of diseases before another class, ask more general questions before specific ones etc.) which is also important in teaching. MYCIN rule base structure was, hence, significantly modified to make the problem solving strategy explicit and accessible to the teaching module in GUIDON. Clancey has continued related work resulting in a recent publication which attempts to unify the various knowledge representation and inference mechanisms [Clancey, 1992].

The concept of an expert system shell came out of the MYCIN work. As part of a thesis, the MYCIN system was *emptied* of its bacteriology knowledge and generalised to become EMYCIN (which stands for Empty MYCIN) [van Melle et al, 1984]. The shell has been used to develop expert systems even in areas totally unrelated to medicine. Some examples are SACON [Bennett et al, 1978] which is a consultant for selecting parameters in a computer program for structural analysis and PUFF [Kunz et al, 1978] which is a system for interpreting pulmonary function test results. Expert system shells are widely used today. Even systems which will eventually be coded in a programming language are often first prototyped using an expert system shell.

Despite the successful validation, MYCIN continues to be an academic system. There are many reasons for this. Partly, in the medical domain one hesitates to entrust the task of diagnosis to a machine – the risks are too high. Secondly, the question of accountability arises. Who is to be blamed if the system makes a wrong diagnosis and the patient suffers because of that? Medical diagnosis is, therefore, one context where the ability to provide explanations become extremely important. Another factor is the narrow domain in which MYCIN works. MYCIN's domain includes only bacterial blood infections, which is a very small part of clinical medicine and it knows nothing about the other aspects of clinical medicine. Even worse, MYCIN assumes that every patient suffers from a bacterial infection!

7.4 XCON

References: [McDermott, 1981], [McDermott, 1982], [McDermott, 1984]

XCON has been called the biggest success story in the area of expert systems. It was a success both in terms of economic returns on the investment as well as being a good expert system application. XCON, initially called R1, was developed jointly by Digital Equipment Corporation (DEC) and Carnegie Mellon University (CMU). The main architect of the system was John McDermott from CMU. XCON is one of the few expert systems whose development has been thoroughly documented. [McDermott, 1981] provides a detailed account of how expert system technology was gradually accepted by DEC.

Unlike most computer vendors, DEC allows each customer to have a unique computer system, by making available a range of options. For example, the VAX 11/780 systems alone had a set of around 5000 components for the customers to choose from. The customer comes with a partial specification. An engineer would sit with him, fill in the missing components, check compatibility of the various components in the order and plan a layout of the system. This was a time consuming process and the layout often had problems due to oversight, etc.

XCON's task was to automate this process. The work on XCON started in 1979 and is still going on. In 1981 DEC installed XCON on their systems and started using it. It is quoted that DEC earns its net investment in the system, every few months (in terms of money) by using XCON. The quick and more reliable processing of orders has also helped increase customer satisfaction.

XCON's outputs include additions or modifications to the order along with explanations, detailed specification based on the user's high level specification, spatial layout of the units in a given room, detailed layout of every unit, power consumption of and heat produced by every unit, unused capacity of memory and buses, exact cable lengths required for various inter-connections, AC and DC loads on the buses, and addresses of devices on the buses. This output is a much more detailed output compared to what human experts generate and is generated in about 10% of the time that they require.

XCON's has been a story of continuous growth. DEC used XCON only for their VAX 11/780 systems to start with. Gradually, the knowledge base was expanded to cover other systems also and is being constantly kept up to date. XCON has been implemented in OPS, a pure production system language. As the system complexity increased, revisions were carried out and more efficient OPS implementations were used (See Section 2.7 for a brief description of OPS). Currently XCON has about four thousand rules and is one of the few very large expert systems in existence.

XCON has two types of knowledge: the constraint knowledge representing the expertise in the domain and the knowledge of components. The latter is stored as a database and has information such as voltage required, frequency, maximum number of devices that can be supported, power specifications, etc. for every component. Typically each component has about 8 parameters of interest and for the VAX-11 system alone, DEC had about 5000 components. Hence efficient storage and access of this database is important for efficient running of the system.

The expertise, i.e., the configuration constraint knowledge, is coded as OPS production rules. These rules encode various constraints to be adhered to while creating a configuration. Specificity as a conflict resolution strategy has been used to advantage in the implementation (see Chapter 2 for a description of specificity).

The task of configuring computers were broken down into six subtasks:

1. Check the order for omissions and correct mistakes
2. Configure the CPU, arranging components in the CPU and the CPU expansion cabinets
3. Configure the unibus modules, putting boxes into the expansion cabinets and then putting the modules into the boxes
4. Put panels in the unibus expansion cabinets
5. Generate a floor plan, grouping components that must be close together and then laying out the devices in the right order
6. Specify what cables are to be used to connect devices and determine the distances between pairs of components in the layout

Each of these tasks consists of a sequence of subtasks, called *contexts* (for example, assign-power-supply, check-voltage-and-frequency). XCON goes through these contexts in sequence. At each step, it would complete a context by doing all that it can in that context and then switch to the next context. In general, XCON works in an exception handling mode at every point. Typically a pair of rules would look like: If A and B and C then do Action1; If A and B and C and D then do Action2. If in a given state, A, B, C and D are all true, then the second rule is preferred to the first (because of specificity). Most configuration knowledge consists of statements like "If such and such conditions exist do Action1, unless this other condition is also true, in which case do Action2". This is easily modelled using rules with specificity as a conflict resolution strategy.

Actually, XCON's rules for switching contexts are very simple

> **If** current context is C1
> **then** set current context to C2

Every rule in XCON (see example rule below) has an antecedent which checks the current context, apart from checking if the current partial configuration meets the requirements of the rule. Hence any rule that does anything in context C1 will be more specific than the context switching rule given above. The context rule will fire only when there are no other rules applicable in context C1, which indicates that it is time to go to the next context.

Here is one of the rules from XCON, paraphrased in English:

If	the most current active context is assign-power-supply
and	a unibus adapter has been put in a cabinet
and	the position it occupies in the cabinet (i.e., the nexus) is known
and	there is space available in the cabinet for a power supply for that nexus
and	there is an available power supply
and	there is no H7101 regulator available
then	add a H7101 regulator to the order

Only about a third of the knowledge in XCON is used for most cases. The other knowledge is for handling special cases and exceptions.

DEC has ambitious plans to employ knowledge based systems on a very large scale in a fully integrated fashion. A few of the systems at DEC are XSEL – an intelligent assistant to a sales person which uses XCON's knowledge to answer *what-if* questions of a customer, ISA – an intelligent scheduling assistant, IMACS – manufacturing control systems, and XSITE – expert for on-site installation and field service.

7.5 ACE

References: [Rauch-Hindin, 1988], [Versonder et al, 1983] and [Siegfried and Wright, 1986]

With the geographical dispersion of telephone networks, maintenance of lines to customers poses a big problem for the telephone industry. AT&T Bell Labs has developed an expert system ACE (Automated Cable Expertise) to help the manager of telephone network centres in this process. Such network centres receive thousands of customer reports and complaints about the functioning of lines in a day. The maintenance personnel have to analyse these reports in order to diagnose the faulty location in the network. ACE automates this task. An interesting aspect of ACE is that it is a fully automatic system with practically no user involvement at any stage.

AT&T has a database management system which collects all the customer reports and maintenance history. The maintenance personnel has to analyse the output from this package (called CRAS).

The mode of working for ACE is as follows: In the night when the load on the CRAS system is low, ACE is automatically invoked. It dials up the CRAS system and downloads the day's reports. These are analysed by the expert system component of ACE. In the process, if it requires any more data, CRAS is dialled up again and the data is picked up. The result of analysis is prepared as mail messages, which are automatically sent using the UNIX mail system to the management staff. The staff who login in the morning examine the reports and schedule appropriate action.

A good part of the administration of the system is also handled by the system itself. Before the processing starts, two modules called Hygiene and Sanitycheck are invoked which clean up unnecessary files, ensure adequate disk space and other resources, etc.

The expert system component of the ACE is written in OPS4 [1], a production system language. It uses forward chaining for inferencing. Here is a rule translated to English:

If	a range of pairs within a cable have generated a large number of customer reports
and	a majority of the work on those pairs was done in the terminal block
then	look for a common address for those pairs.

The work on the system started around 1980-81 and the system was in field test by the end of 1982. The system runs on a 3B2/300 microcomputer. It takes about 1 to 4 hours for a full run.

7.6 CATS-1

References: [Harmon and King, 1985], [Rauch-Hindin, 1988]

CATS-1 is a computer aided troubleshooting system for locomotives. CATS-1 (also called DELTA) was in real use. The work on it started around 1981 at General Electric (GE), Schenectady, New York and a production version consisting of about 1200 rules was put to use around 1984. The system could handle upto 80 percent of the troubleshooting problems. This system was phased out in 1986 due to various reasons, including the difficulty of updating the knowledge base and improper transfer of the system to users.

General Electric Company's diesel electric locomotives are maintained by the railroads purchasing them. The first level of maintenance and repair are carried out in what are called "running repair shops", the equivalent of fancy petrol stations. Minor repairs (typically requiring less than 2 hours) are done here. Locomotives requiring more complex operations are sent to major repair shops, when they are not able to diagnose the problem. It turns out that quite often locomotives requiring minor repair are sent to major repair shops by these first level shops, where they end up waiting for a long time and also tie up expensive repair machinery. To reduce this problem and increase productivity of the first level shops, GE developed an expert system for troubleshooting these diesel electric locomotives. The system was installed and used at the first level shops.

The main expert for CATS-1 was David Smith who had been with GE for more than 40 years and whose expertise they wanted to capture. The system was written in the FORTH programming language so that it could be made

[1] OPS4 was a predecessor of the OPS5 language discussed in Chapter 2

available to repair shops in an affordable manner. The system ran on a PDP-11/23 desktop minicomputer. It required half a megabyte of memory and 20MB winchester along with graphics and networking capabilities. The networking was used to access failure history and typical features of a given model from GE's main computer system, after taking the model number from the user.

The knowledge in CATS-1 is represented in the form of rules. The system uses both backward chaining and forward chaining as inference procedures. The rules can be classified as If-rules, Iff-rules, When-rules and Meta-rules. The When-rules are used by the forward chainer. The forward chainer reacts to changes in the situation and is invoked every time some additional information is added to the system. This propagates the effects of the new information as far as possible to the other parts of the rule base. The selection of applicable rules in the backward chaining mode is decided by an explicit rule selection strategy. This selection strategy is based on the cost of gathering the additional evidence required by the rule.

Here is a sample rule:

```
IF      EQ [ENGINE SET IDLE]
        EQ [FUEL PRESSURE BELOW NORMAL]
        EQ [FUEL PRESSURE GAUGE USED IN TEST]
        EQ [FUEL PRESSURE GAUGE STATUS OK]
THEN    WRITE [FUEL SYSTEM FAULTY] 1.00
```

The system is capable of handling multiple faults also. But it does it sequentially — i.e., the system will point out one fault and after it is fixed the system should be rerun to see if there are any other problems.

One of the nice aspects of this system is its user interface. Some of the rules in the system are for deciding what help or information is to be provided to the users. The system may need to know if certain sounds expected from the locomotive (say, while starting the engine) are normal. For these kind of things, the system has an audio connection which it can use to play to the user the normal expected sound for comparison. Also to show an inexperienced user how to open up some part and get data or how to carry out a suggested repair, the system can show a video recording of the relevant actions. For the kind of application domain that the system was meant for, this investment in the user interface was absolutely essential.

7.7 Other Systems

Expert Systems have been developed for many application domains varying from standard fault diagnosis type of applications to tax planning and transportation schedule design. In this section, we list some specific domains for which commercial expert systems have been developed. The list has been categorised according to the domain of applications. The list is by no means exhaustive; but is only meant to give the reader a feel for the scope of expert systems technology.

The case-studies given in later chapters, will examine some of these domains in more detail.

Management Applications

* Scheduling and person-loading of software development projects
* Preparation of proposals for purchase of capital equipment
* Guiding managers in shaping the performance of employees

Financial Applications

* Assistance with business planning
* Analysis of loan applications and design of loan packages
* Risk evaluation in insurance applications to determine pricing
* Investment advisor for people
* Advice on sales tax status of financial transactions

Office Automation Applications

* Letter of credit advisor
* Determining security classification for documents
* Deciding if an applicant should be called for an interview

Manufacturing Applications

* Design of brushes for electric motors
* Advising coal mine operators on dust control
* Planning schedules for work

Equipment Maintenance Applications

* Telephone cable maintenance advisor
* Locomotive troubleshooting
* Assisting steam generator design
* Diagnosing preventive maintenance needs for large central air conditioning systems
* Vibration problems in large turbo machinery

Computer Applications

* Cable configuration for computer systems
* Computer peripheral diagnosis
* Direct access storage device diagnosis

Transportation Applications

* Air cargo ship load configuration
* Advisor on how to handle, label and ship hazardous chemicals
* Monitoring control on space shuttle flights

Other Applications

* Irrigation advice to peanut farmers
* Identify suitable planting equipment
* Drug interaction analysis
* ECG interpretation assistant
* Solving mathematical problems symbolically

Summary

- ESs are being used by many industries to increase productivity by providing automated assistance. Many of these applications are difficult to solve using conventional programs. If-then rules is the most popular knowledge representation formalism, so far. This chapter describes some expert systems in use.

- DENDRAL: One of the first expert systems. The task was to help chemists in determining the molecular structure of an organic compound, from mass spectrometer data.

- PROSPECTOR: The task was to help geologists in early stages of investigating a site for ore-grade deposits. Uses a variant of Bayesian reasoning for handling uncertainty. Provides mixed mode interaction and a restricted natural language interface.

- MYCIN: A seminal expert system. The task was diagnosis and therapy selection for bacterial infections of blood. Introduced the notion of certainty factors, the idea of an expert system shell and putting a knowledge base to multiple uses (TEIRESIAS, GUIDON etc.).

- XCON: Most successful expert system. The task was to validate customer specifications of computer systems and design the complete configuration of the system. Uses a forward chaining model and is developed using the production system language OPS.

- ACE: An application in the area of engineering. The system helps telephone network managers to identify faults in the network, by analysing the customer reports and maintenance history records. The system is in use at the AT&T Bell Labs.

- CATS-1: The task was troubleshooting diesel electric locomotives. The system makes good use of audio and video in the user interface. The system was in real use for sometime before it was phased out.

8. Knowledge Engineering

Who creates expert systems? What is the work involved in creating them? Who defines the scope of the expert system (ES)? Who decides that the system is good enough for regular use? This chapter will deal with these, and other related issues.

8.1 Creating Knowledge Bases

Creating a knowledge base differs in a very significant way from creating conventional computer software. To begin with, creating a knowledge base is quite unlike any form of coding. One does not create a procedural solution to a problem, in creating a knowledge base. One captures knowledge in a non-procedural (declarative) form, usually, for use with an expert system shell. But on the other hand, the task is very demanding in its own right, requiring far more in craftsmanship than writing an essay on the topic on which the knowledge base is going to be created. This is because the knowledge has to be represented in a precise and unambiguous way. The professional who handles this task is the *knowledge engineer* (KE). He works closely with the subject specialist, usually called the *domain expert* (DE). The DE has the knowledge expected to be encapsulated in the proposed ES.

Maintaining a knowledge base is easier than maintaining conventional software. Items of the knowledge base capture domain knowledge in a relatively modular way. So, one can often change a single rule without too much understanding of the rest of the system, and still get the desired improvement. Similarly, one might add a few rules equally easily.

However, the work of KE is demanding in many respects. He has to:

- identify all relevant knowledge in the chosen domain
- be familiar with the specifics of the knowledge representation language used
- understand the techniques used by the system for reasoning under uncertainty
- collect items of information and knowledge from the DE, and from relevant literature
- represent this knowledge using the knowledge representation language

- work closely with the domain expert to refine the knowledge base after getting feedback from trial runs of the system
- finally, validate the system through well designed tests

8.2 What is to be Created?

A typical ES development environment consists of the following modules:

- knowledge acquisition module
- knowledge-base refinement module
- problem solving module
- explanation module

The knowledge acquisition module enables the system to acquire knowledge from external sources. The acquisition of knowledge by a system could involve manual work and supervision to a varying degree. At one extreme, the module could, for example, just be a good rule-editor for use by the KE in creating new rules and improving existing rules. At the other extreme, it might be far more sophisticated. For example, it could be a component handling automatic examination of data, and creating new items of knowledge. The KE might have to examine and decide if these new items are good enough to be incorporated into the system, and edit them if necessary. But a fair amount of automated data processing, for knowledge acquisition, is possible in many domains.

Consider the following rules from the expert system Hruday (see Chapter 11 for details about Hruday):

```
rule (ihr10, IH_risk, true, 0.10) :-
    gt(age, 50.0).
rule (sd3, suspected_disease, RHD, 0.18) :-
    is(card_symp_present, true),
    gt(age, 10.0),
    le(age, 40.0).
rule (sd6, suspected_disease, CHD, 0.18) :-
    is(card_symp_present, true),
    lt(age, 40.0).
```

The first rule says that the patient's age being greater than 50 contributes 10% to belief in the hypothesis that he runs the risk of suffering from Ischaemic Heart (IH_risk) Disease. The second rule, sd2, contributes 18% to the belief that the disease involved is Rheumatic Heart Disease (RHD) if all the three conditions given are satisfied. Age of the patient is an important contributor in the "differential diagnosis" spelt out in this rule. The third rule, sd5, is also a similar rule, using age of the patient as a clue in diagnosing Congenital Heart Disease (CHD). This rule is based on the fact that CHD is a condition present from birth, and usually detected at a young age. In comparison, Ischaemic Heart Disease is relatively rare among those below 50.

How did the age specific information get derived? In the case of Hruday, this information came from books. A knowledge acquisition system working on a large medical database can create rules like these, using constants and confidence factors which are more accurate.

The refinement module facilitates refinement of the knowledge base. For instance, in Vidwan, the rule editor enables manual editing of the rules. More advanced refinement modules can process recorded case histories, and the conclusions a given ES came to on these case histories, to offer suggestions for improvement of rules. They could automate part of the work involved in the efforts. Such systems are described in Chapter 9.

The problem solving module applies available knowledge to the problem at hand. In the case of rule based expert systems, this module is the *inference engine* which applies appropriate rules, to perform the basic problem solving task.

The explanation module provides the user with explanations of the behaviour of the ES. Most expert systems provide two essential forms of explanation:

- **Why** — Why is a certain question being asked by the ES?
- **How** — How was a certain conclusion reached by the ES?

Producing answers to *Why* and *How* questions in a mechanical fashion does not complete the commitment of an ES to provide meaningful explanations. Very often, it is necessary to plan for explanations from the very beginning of the design of the ES. The way knowledge is represented has a big impact on the way explanations are produced. So, knowledge representation has to be planned with explanations also in mind. For more issues related to explanations, refer to Chapter 6.

8.3 Contrast with Software Engineering

The stages of typical software development are:

- defining requirements
- specifying software to be created
- designing the software
- implementing the software
- testing and refining the software

Implementation of most conventional software is carried out in procedural programming languages, such as C, Pascal or FORTRAN. This involves writing out a sequence of actions to be carried out by the machine. The creation of a knowledge base is quite different, involving the creation of declarative representations of relevant knowledge. The inference engine uses its built-in problem-solving procedure, using the knowledge base created by us, to solve the problem at hand. The work is far more *problem-oriented* than *machine-oriented*.

Knowledge base creation needs its own methodology. Usually, knowledge based systems are *not* created by developing well-defined program modules in relative isolation and then integrating them. It involves viewing of the problem in its entirety. Starting with a very limited system for coping with a part of the problem, one gradually adds to it rule after rule, to increase the scope of its coverage. Simultaneously, one refines the rules as and when errors, conflicts, or deficiencies are detected. Thus, incremental acquisition and representation of knowledge is the central element of the methodology.

8.4 Knowledge Engineer and Domain Expert

The knowledge engineer handles the equivalent of what the systems analyst and the programmer handle in conventional software. However, it is far easier for a specialist, such as a medical doctor or a mechanical engineer, to become an effective KE than it is for him to become an effective C programmer. The KE needs to be knowledgeable about computers, artificial intelligence and expert systems. In particular, he should have adequate familiarity with the knowledge representation language associated with the tool (or shell) he is using. He should have used a number of expert systems, so that he has a feel for the good and bad features of these systems. He should have experience in building a series of expert systems of gradually increasing complexity. He must have a good understanding of what makes a good user interface, and what does not. He should usually have good communication skills, to ensure that the sentences used by the system being built are succinct, clear, precise and elegant.

The DE is an experienced and knowledgeable professional whose specialist knowledge is to be captured, to the extent possible, in the expert system to be created. He provides the main source of information to the KE, giving the KE access to his own expertise in a variety of ways. Some of the ways in which he can do this are given below:

- briefings
- interviews
- demonstrations of problem solving
- identifying relevant information sources such as manuals
- creating good test cases with which one can test the evolving ES
- critically analysing the output of the prototype ES

Sometimes, a short training programme relevant to the task of the proposed ES may be available. The KE might sit through such a programme, to identify the knowledge being transmitted to trainees. For instance, the KE will find it useful to attend a short course in giving first-aid to poison victims, before writing an expert system useful in such work.

The DE has expertise which has to be *externalised* during the creation of the ES. He is normally not conscious of the process of problem solving that he uses to handle the task. He cannot easily identify the mechanisms used in his own problem-solving. The KE works with him, interviewing him, asking him questions as he goes on with his work, proposing rules that seem to capture some knowledge, and so on. He may ask the DE to think aloud while solving a problem, or ask him to justify why he handled a problem in a particular way. The process of identifying the knowledge that is to be incorporated in the ES, and putting it in the right external form is called *knowledge elicitation* .

Knowledge elicitation cannot be automated to a high degree at the current level of the technology. It continues to be handled usually manually, and is subject to errors of judgement on the part of the KE.

8.5 Phases of Knowledge Engineering

The KE goes through a series of phases in creating a knowledge base:

- identifying the domain to be covered by the ES
- identifying the concepts to be incorporated in the ES
- "formalising" the concepts
- implementing the knowledge base
- testing the implementation
- refining the implementation

The creation of the knowledge base is not a simple one-way trip down this path! Often, one learns something at a certain phase, to go back to the previous phase and rectify an error at that stage. Such feedback is essential for the creation of any complex expert system. These phases of knowledge engineering will be described in the following sections.

8.5.1 Domain Identification

Domain identification requires that the KE should:

- acquire adequate familiarity with the field
- decide that the task is appropriate for using an ES
- exercise his judgement to define the scope of the ES to be built
- decide what hardware and software will be required to implement the ES

- estimate the effort involved in creating the ES
- estimate the number of professionals to be involved and the time to be taken in creating the ES
- create a rapid prototype of the ES, to get a better estimate of the effort involved

Creating a *rapid prototype* offers a number of advantages. It lets the domain expert see the form in which the knowledge is going to be represented. It indicates likely problems that will be encountered in creating the ES, depending upon how amenable the task is to the ES approach, and gives a feel for the complexity of the effort involved.

The process of prototyping also indicates how suitable a chosen knowledge representation language is for the task at hand. It gives the KE a feel for the level of expertise available to him through the domain expert and other sources.

Very often, the process of prototyping may prompt the KE and DE to change the scope of the proposed ES, in the light of all the experience it provides.

8.5.2 Domain Conceptualisation

The KE ends up being a bit of a DE in the process of creating an ES! He acquires relevant knowledge, including concepts, domain-specific relations, the nature of these relations, and relations between concepts.

For example, in an ES for the insurance industry, one would have concepts and relations such as the following:

- Concepts: customer, old-customer, new-customer, good-risk, medium-risk, high-risk, premium, length-of-coverage, type-of-coverage, etc.
- Relations: premium(type-of-customer, risk),
  bonus(customer-record, age-of-vehicle, type-of-vehicle), etc.

8.5.3 Formalising Concepts

When a KE represents knowledge in a systematic manner, he is in a sense *formalising* knowledge. The knowledge is put into structures that the ES can use. The ES does not "know" the meaning of the structures it manipulates. It uses only the *form* of the structures it encounters. In that sense, the work of a KE involves formalisation. However, this does not really correspond to the formalisation that a logician or a mathematician employs.

The structures used to represent knowledge include the following:

- hierarchy
- taxonomy
- if-then form

Many of the ES tasks involve classification of entities: bacteria, diseases, gears, IC chips, and so on. Sometimes, it is appropriate to arrange them in a hierarchy using a tree-like diagram, to show what entities form a sub-group, and how sub-groups can be combined to form higher level sub-groups and so on. Taxonomies handle more complex classification information. If-then forms contain information about how one can take steps of logical reasoning, proceeding from *antecedents* to *conclusions*. Usually, If-then forms can directly be converted to working rules in rule based expert systems.

The precise representation of knowledge will require a knowledge representation language providing for rules, frames, semantic nets or logical propositions. In certain domains, however, one may wish to use knowledge representation languages which provide something very specific to the domain. For instance, in a domain in the area of chemistry or chemical engineering, one may use a machine-manipulable expression for the chemical formula of a molecule. Such a specialized knowledge representation language will be valuable for capturing important knowledge in the domain.

8.5.4 Testing and Refinement

The ES under development has to be tested, refined and validated. The best technique would be to create a machine readable collection of data covering an adequate number of test cases. The number of cases to be used would depend upon the complexity of the domain. The cases, as a whole, should be representative of the types of cases the ES will have to handle.

One can get the DE to look through the cases and record his solutions. The same cases can then be processed by the ES and the results can be compared. This will help identify missing knowledge, errors in knowledge represented in the system, and the need for refinement of knowledge represented. For instance, one may find weaknesses remediable through the addition of new if-then rules. Or, one may find the need to increase or reduce certainty factors. One may find the need to split a single if-then rule to handle two different sub-cases differently.

In general, one is pitting the ES against the DE's level of performance. A very high standard is to compare the performance of the ES against the judgement of a panel of, say, five DEs. Interestingly, medical expert systems exist which perform very well in the light of the judgement of a panel of experts. They often outperform individual DEs!

How good or bad an ES is, obviously, depends upon the nature of the knowledge base used. The simplest remedy to poor performance is improvement of the knowledge base.

Two concepts very relevant to testing an ES are *false positives* and *false negatives.* For example, if an ES diagnoses a patient as having a heart disease when he does not have one, we have a case of a *false positive*. In another case, the ES may look at the symptoms of a person with heart disease and declare him to be free of the disease. This is a *false negative.*

We can define false positives and false negatives as follows:

- *False positive* — inferring a conclusion *not* called for by the evidence

- *False negative* — *not* inferring a conclusion called for by the evidence

False positives can often be eliminated by reducing the CF associated with some relevant rule. In some cases, false positives can be eliminated by making the antecedent of a rule more restrictive. As an example, consider false positives in the case of a heart disease advisor. Imagine that certain patients who report being short of breath get wrongly identified as having heart disease. On investigation one might find that one has to rule out those with asthma in applying this rule. The shortness of breath might have been due to asthma rather than to heart disease. Therefore, the antecedent might be made more specific, adding the no-asthma requirement to eliminate wrong conclusions.

Similarly, false negatives can be reduced, in many cases, by increasing the CFs of the relevant rules. Alternately, one might have to make the antecedent of the rule less restrictive.

A false negative could indicate a missing rule. A false positive may indicate an erroneous rule (see Chapter 9 for further discussion of this topic).

8.6 Tools for Knowledge Engineering

EMYCIN incorporated a KE tool named TEIRESIAS [Davis, 1979], a knowledge editor. TEIRESIAS used the concept of *Rule Models* . For instance, there is a rule model applicable to rules which identify an organism causing a disease. The rule model indicates that such rules usually refer to the type of infection and to the site of the infection. If the KE tries to create a rule of this kind, TEIRESIAS would help him by verifying that the new rule meets this expectation. Otherwise, the KE would be informed of the violation of this expectation. TEIRESIAS does not use any domain knowledge and is domain independent. It only uses the *syntactic* form of rules to come to its conclusions.

SEEK [Politakis and Weiss, 1984] and SEEK2 [Ginsberg, 1988] offer advice on refinement of rules. SEEK2 is described in Chapter 9.

XPLAIN [Swartout, 1983] makes it easy to get good explanations of ES behaviour. Special provisions for handling explanations are included in the design of XPLAIN. In the XPLAIN methodology, *a domain model* containing domain-specific facts, causal information and taxonomic information, is first created. The domain model is merely declarative. XPLAIN uses automatic programming techniques to create ES rules from the domain model. This makes it easy for the ES to be equipped to provide simple and effective explanations of its behaviour when these are required.

In the case studies at the end of the book, we examine some of the knowledge engineering issues and discuss the structuring of knowledge bases with respect to specific application domains.

Summary

- Creating an expert system is significantly different from creating a conventional program, mainly because the knowledge of the domain has to be captured in a declarative form.

- The knowledge engineer is the person who talks to the domain expert, acquires information about the domain, and structures it in a form which can be used by the ES.

- An expert system development environment consists of the following modules: knowledge acquisition module, knowledge-base refinement module, problem-solving module and explanation module.

- The knowledge engineer can acquire knowledge from the domain expert in a variety of ways such as: briefings by the domain expert, interviews, looking at the expert solving real problems, looking at published material, etc.

- Creation of prototypes is useful while developing an expert system.

- The knowledge engineer has to have a basic understanding of the concepts and terminology for the domain for which he is creating an expert system.

- After knowledge is acquired from the expert, it has to be cast in a representation scheme the system can directly use.

- Problems that sometimes occur in rule bases are: circular rules, conflicting rules and redundant rules.

- Testing and refinement are important phases in the development of an expert system.

- Tools are now becoming available to support the task of Knowledge Engineering.

Exercises

1. You have to design an expert system for troubleshooting a television receiver. You are going to have your first meeting with the domain expert. You have to plan a set of questions, answers to which will help you better understand the domain. Create a set of questions you should ask the domain expert in this meeting.

2. You have 10 test cases, in which the inputs to an expert system and the appropriate diagnoses are given. How would you test the expert system and determine rules which have to be refined, based on the diagnoses which the expert system gives and the correct diagnoses you have been given?

9. Rule Base Verification, Refinement and Validation

9.1 Introduction

Verification, refinement and validation of expert systems are important parts of the development cycle of an expert system. In fact, these take a bulk of the development time for most typical domains.

The first prototype of an expert system does not take much of the overall development time. However the first prototype often has problems like:

- The knowledge it has is incomplete
- Some of the knowledge is incorrect

If the system's knowledge is incomplete it may not be able to derive some information it is expected to. It may also lead to incorrect conclusions. If the knowledge is incorrect, the system could exhibit strange behaviour – working for some cases and getting other cases totally wrong. Systematic verification and validation is then required in order to correct these problems. After you have corrected these problems in a rule base you may have to verify and validate the rule base again to see if there are any other problems. Therefore verification and validation are not just done once but have to be done more or less continuously after the first prototype of an expert system is developed.

We first define the terms verification, refinement and validation of expert systems. Verification of an expert system deals with determining whether the system was developed correctly, that is, whether the system's knowledge satisfactorily represents the domain knowledge.

Refinement of a rule based system deals with improving the performance of the system. Improvement of performance could involve correcting bugs in the rules in the system, dropping or adding conditions in a rule, changing certainty factors in the rules, etc.

Validation of expert systems deals with trying to determine whether the system does what it was intended to, and at an adequate level of accuracy. This is obviously related to refinement. While validating an expert system, if you find that it does not meet expectations, it has to be refined.

In this chapter we will try to examine some of the issues related to these in the context of rule based expert systems.

9.2 Expert Systems vs Conventional Programs

Programmers may ask how the issues mentioned above are different for expert systems as compared to conventional programs. One major difference is that most conventional programs have underlying algorithms on which they are based whereas expert systems are inherently non-algorithmic. However there are some similarities in the way one would test the two classes of systems.

The normal way in which computer programs are tested is to provide them with a set of test cases. If there are any problems with the outputs for these test cases, the user tries to determine which part of the program could have lead to the problem. He will gradually converge to the erroneous part of the program based on its functional behaviour.

A similar sort of method can be used for expert systems. However, in general, the more complex a system is the more number of test cases it will require. Some expert system tools in fact allow you to store cases run on an expert system and rerun them whenever you want.

The explanation facilities, which expert system shells provide, are in a way equivalent to the debugging facilities provided in programming environments. The explanation facilities allow one to examine the reasoning strategy used by the expert system. This examination can lead to the detection of reasons for the erroneous behaviour of the system (buggy rules, missing rules etc.).

9.3 Verifying Expert Systems

As mentioned earlier, verifying an expert system is essentially verifying the knowledge which has been represented in the knowledge base of the expert system. We can normally assume that one is implementing an expert system on a well-tested expert system shell. In that case the problem is normally because of the domain knowledge and not because of problems with the expert system shell.

The type of problems which are common to rule based systems are:

- Redundant Rules
- Conflicting Rules
- Subsumed Rules
- Circular Rules
- Unnecessary Antecedents
- Unreferenced Attribute Values
- Illegal Attribute Values
- Unreachable Conclusions
- Dead-end **If** conditions

- Dead-end Goals

We will first describe each of these in the context of rule based systems which do not use certainty factors, and then see how the inclusion of certainty factors affects them. The ideas described in this section have been implemented in programs named ARC (ART Rule Checker)[Nguyen, 1987] and CHECK [Perkins et al, 1989]. The CHECK program was a verification tool which was implemented for the Lockheed expert system shell, which is a rule based expert system shell. The ARC tool comes along with the expert system tool ART (Automated Reasoning Tool).

9.3.1 Redundant Rules

Two rules are redundant if they succeed in the same situation and have the same conclusions. Examples of rules [1] which are redundant are given below:

rule(r1, can_fly, true) :- is(bird, true), is(ostrich, false).
rule(r2, can_fly, true) :- is(ostrich, false), is(bird, true).

These two rules have the same conditions, ordered differently.

Redundant rules cause serious problems in systems using certainty factors. A set of redundant rules would make the confidence in the result be counted more than once. In the worst case, a consequent which should have had a certainty below the threshold could be pushed above the threshold making it true.

Examples of redundant rules using certainty factors are:

rule(r3, jaundice, true, 0.12) :- is(fever, true).
rule(r4, jaundice, true, 0.12) :- is(fever, true).

These two rules are the same and therefore one of them is redundant. Note that they could have occurred at different places in a rule base and may have been overlooked by the knowledge engineer.

Fever is a symptom for jaundice; so, if a person has fever, it is weakly suggestive that he has jaundice. But there could also be other reasons why he has fever. Fever by itself is not sufficient to conclude jaundice; that is why the certainty factor for jaundice was given as less than 0.2. But because of the redundant rule, if a person has fever, both rules will be satisfied and the system would conclude that a person has jaundice, as the certainty of jaundice would go beyond 0.2 (Refer to Chapter 5 details on how certainty factors are used). This is obviously an invalid conclusion.

9.3.2 Conflicting Rules

Two rules are conflicting if they succeed in the same situation but lead to contradictory conclusions. Examples of conflicting rules are:

rule(r1, can_fly, true) :- is(bird, true), is(ostrich, false).
rule(r5, can_fly, false) :- is(bird, true), is(ostrich, false).

[1] In this chapter, for rules without certainty factors we use the Vidwan rule format without the fourth argument in the consequent. See Appendix A for details of Vidwan syntax.

In fact rule *r5* is not correct.

In systems with certainty factors, often rules with the same antecedents may lead to different conclusions with different certainty factors.

Consider the rules:

rule(r3, jaundice, true, 0.12) :- is(fever, true).
rule(r6, jaundice, false, 0.10) :- is(fever, true).

These two rules have the same condition, but have different certainties for the conclusion *jaundice*. One of these rules should therefore be discarded.

On the other hand the set of rules:

rule(r3, jaundice, true, 0.12) :- is(fever, true).
rule(r7, typhoid, true, 0.10) :- is(fever, true).

are perfectly reasonable rules to have in a system. Even though both of them have the same condition, they update the certainties for two different conclusions *jaundice* and *typhoid*. This is reasonable because the same symptom could lead to a number of different conclusions.

9.3.3 Subsumed Rules

One rule subsumes another if the conditions in one rule are a subset of the conditions in the other and both the rules have the same conclusions. Examples of one rule subsuming the other is given below:

rule(r1, can_fly, true) :- is(bird, true), is(ostrich, false).
rule(r8, can_fly, true) :- is(bird, true).

The rule *r8* subsumes rule *r1* because it has only one of the conditions of *r1* but has the same conclusions.

Systems with certainty factors often use subsumption in a meaningful way. In these systems, the more general rule often assigns an initial certainty factor. The more restrictive rule (the subsumed rule), if applicable, will then increase or decrease the certainty of the conclusion.

For example:

rule(r3, jaundice, true, 0.12) :- is(fever, true).
rule(r9, jaundice, true, 0.50) :- is(fever, true), is(eyes, yellow).

Out of these two rules the first rule subsumes the second rule. So one may believe that the second rule is not required. This may not however be true. If a person has fever, you can assume that the disease could be jaundice with some certainty. When the second rule is also satisfied you increase your confidence in the conclusion. In this example, the fact that the person has yellow eyes more or less confirms that he has jaundice and therefore the certainty goes well beyond the threshold.

9.3.4 Unnecessary Antecedents

Two rules have unnecessary antecedents if one rule has a condition and the other rule has the negation of the same condition and the two rules lead to the same conclusion. Consider the rules:

rule(r1, can_fly, true) :- is(bird, true), is(ostrich, false).
rule(r1a, can_fly, true) :- is(bird, true), is(ostrich, true).

These two rules can be substituted by a single rule:

rule(r1, can_fly, true) :- is(bird, true).

Again in systems with certainty factors, rules with unnecessary antecedents may be necessary if the conclusions have different CFs.

For example consider the rules:

rule(r9, jaundice, true, 0.50) :- is(fever, true), is(eyes, yellow).
rule(r10, jaundice, true, 0.18) :-
isnot(fever, true), is(eyes, yellow).

Both these rules can be substituted by a single rule:

rule(r11, jaundice, true, 0.18) :- is(eyes, yellow).

In these two rules the conclusion is the same, but the first antecedent in the second rule is the negation of the first antecedent in the first rule. Therefore the two rules can be substituted by a single rule.

On the other hand the following set of rules are reasonable:

rule(r9, jaundice, true, 0.50) :- is(fever, true), is(eyes, yellow).
rule(r12, jaundice, true, -0.2) :- isnot(fever, true), is(eyes, yellow).

In the second rule the conclusion *jaundice* is concluded with negative certainty (the possibility of jaundice is therefore decreased), if the person does not have fever. So even though both rules conclude *jaundice* and one rule has an antecedent which is the negation of an antecedent in the other rule, the two rules are still correct, because the certainty factor is positive in one case and negative in the other.

9.3.5 Circular Rules

A set of rules are circular if they are of the following form:

If A **then** B
If B **then** C
If C **then** A

If you use backward chaining here, and are trying to determine A, then you would need to determine C. To determine C you need to determine B. But to determine B you need to determine A. That means in order to determine A you need to determine A. The rules are therefore circular.

Most systems will loop or show unpredictable behaviour in the presence of circular rules. Circular rules can also occur in systems which support certainty factors. However, circular rules could lead to decreasing the certainty until it goes down below a certain threshold, thus avoiding the apparent circularity.

The CHECK system [Perkins et al, 1989] produces a dependency chart which is useful for detection of a set of circular rules. The dependency chart is a matrix with the columns and rows named with unique identifiers for the rules in the system. An asterisk ($*$) in an element of the matrix indicates that an antecedent in the rule corresponding to that row is concluded by a consequent of the rule corresponding to the column. In a dependency matrix if the intersection of the nth row and nth column is a $*$, the rule is a circular rule. On the other hand if there is a $*$ in the mth row and nth column and another $*$ in the nth row and mth column, these two rules together form a set of circular rules.

Consider the set of potentially circular rules given below:

rule(r13, A, true, 0.2) :- is(B, true).
rule(r14, B, true, 0.2) :- is(A, true).
rule(r15, C, true, 0.4) :- is(A, true).

The dependency chart for this set of rules is as follows:

	r13	r14	r15
r13		$*$	
r14	$*$		
r15	$*$		

Since there is a $*$ in the element corresponding to the *r13* row and *r14* column and another $*$ in the element corresponding to the *r14* row and *r13* column, the two rules *r13* and *r14* are potential circular rules. The same idea could be extended to determine whether a set of n rules are circular or not – but this would be obviously harder to find out.

The dependency chart is therefore a simple visual mechanism to determine whether a set of rules are circular or not. One can consider segmenting a rule base into groups of rules which are related to each other, and creating dependency charts for each group to detect possible circularities in the system.

9.3.6 Unreferenced Attribute Values

Each multi-valued attribute in a rule has a set of possible values it can take. A complete rule base would require that every value of an intermediate attribute occurs in the condition part of some rule. If an attribute value is not used in any rule, then that attribute value is unreferenced.

Consider the attribute *weather* which has *cold*, *normal* and *hot* as possible values. Suppose that there are three rules for temperature:

```
rule(r16, weather, cold, 1.0) :- lt(temperature, 10).
rule(r17, weather, normal, 1.0) :-
        ge(temperature, 10),
        le(temperature, 30).
rule(r18, weather, hot, 1.0) :- gt(temperature, 30).
```

Now suppose that there are no rules with the condition *is(weather, hot)*. This could mean one of three things. The rule base designer may have mistyped the attribute value for weather. Or he may have intentionally added the rule r18 so that the system could be extended later. Or there is a missing rule which should use this condition. In some of these cases the system may come out with incorrect results.

9.3.7 Illegal Attribute Values

This problem is related to that of unreferenced attribute values. In this case, the value is not among the set of values the attribute can take and is therefore illegal. Such problems could be because the value of an attribute is misspelt.

Suppose instead of the rule r18 we had given above, we had:

rule(r18, weather, hto, 1.0) :- gt(temperature, 30).

Then obviously the attribute value *hto* for weather is illegal. The designer probably wanted to put the value *hot* but may have made a typing mistake. Most expert system shells would be able to detect this, by checking if the value *hto* is a member of the valid set of values (cold, normal,hot) for weather, which was pre-defined for that attribute.

9.3.8 Unreachable Conclusions

In a backward chaining system the conclusion of a rule should match part of the user goal or match the **If** condition of some other rule. Otherwise, this conclusion is unreachable. This problem will not occur in forward chaining systems.

Detecting unreachable conclusions in a rule set with certainty factors is more complex. A conclusion in a rule could be unreachable even if its **If** condition matches the conclusion of some other rule. This is because the condition may have a certainty always less than the threshold required for it to be satisfied. For example consider the rules:

rule(r19, B, true, 0.15) :- is(A, true).
rule(r20, C, true, 0.8) :- is(B, true).

Suppose rule *r19* is the only rule which has the attribute B as conclusion. Even if attribute A is absolutely certain, B cannot have a certainty greater than 0.15. Rule *r20* will never succeed since the certainty of the condition B will never be greater than the threshold. Therefore, C is an unreachable conclusion.

9.3.9 Dead-end If Conditions and Dead-end Goals

If an **If** condition of a rule cannot be evaluated either by asking the user for the value or by evaluating some other rule, then the condition is a dead-end condition. The rule will never fire.

Similarly if a goal cannot be derived, either by asking the user or by evaluating some other rule, the goal is a dead-end goal.

Detecting dead-end **If** conditions in systems which support certainty factors is much more difficult. This is because a goal is a dead-end goal if it can never be concluded with a certainty greater than the threshold in a system. It is possible that successful combination of intermediate goals could lead to a reduction in the certainties until it falls below the threshold.

For example consider the following rules:

rule(r21, B, true, 0.4) :- is(A, true).
rule(r22, C, true, 0.7) :- is(B, true).

rule(r23, D, true, 0.7) :- is(C, true).

Suppose A is known with full certainty (CF = 1.0), the certainty for B would be 0.4, the certainty for C 0.28 and the certainty for D 0.196. Since the certainty for D is less than the threshold of 0.2, D is a dead-end goal. If the attribute D was not a goal but was an **If** condition of some other rule, then this would lead to a dead-end **If** condition, if D is not askable and there are no other rules which conclude D.

9.4 Rule Base Refinement

The problems we have described can be detected by examining the rule base. Another way of finding problems in a rule base is to use case histories [Sasikumar et al, 1988].

In a number of domains, a large number of cases of the types of problems the system should handle and their results, are known. After developing an expert system would it be possible to run the cases through the system and see how they perform? Would it be possible to use the results of this run to refine the rule base?

The answer to both these questions is yes. In fact a system named SEEK2 has been developed which does this [Ginsberg et al, 1987]. SEEK2 was an enhancement of an earlier system named SEEK [Politakis, 1985]. SEEK2 takes the rule base, examines its performance on a set of cases and based on this, suggests changes which can be made to the rule base to improve its performance. The suggestions are in the form of specifying rules which can be modified and specifying how they can be modified.

As in conventional programs, expert systems are very seldom *correct* as soon as they are created. One needs to go through a process of testing and refinement in order to improve the performance of a system. Refinement of knowledge normally refers to fine tuning an expert system. This could be by adding or deleting a condition/conclusion to a rule, or increasing or decreasing the certainty associated with a rule.

We will not be able to give a description of exactly how SEEK2 works because of lack of space. However we will try to outline the general principles which it uses. This description should give some idea of how rule bases can be analyzed using a set of cases. Tools like SEEK2 are likely to be incorporated into high end expert system shells in the future.

Before we go ahead we need to know some terminology. The first two terms we need are *false positive* and *false negative*. For a case, if a system was supposed to come to a conclusion A, but did not, then the conclusion is a false negative. On the other hand, for a case, if the system was not supposed to come to a conclusion A but came to it, then the conclusion is a false positive.

The other two terms we need to know about are *generalization* and *specialization* of rules. A rule generalization is modifying a rule so that it is easier to satisfy. This can be normally done by either deleting or relaxing a condition of the rule or by increasing the certainty factor for the conclusion of a rule. A rule specialization is the opposite of rule generalization. This is modifying a rule so that it is harder to satisfy. This can be normally done by either adding or strengthening a condition of the rule or by reducing the certainty factor of the conclusion in the rule.

Now that we have defined the terms, let us look at the strategy SEEK2 uses. We will illustrate this using an example taken from [Ginsberg et al, 1987]. Consider a rheumatology knowledge base dealing with the two final diagnoses Systemic Lupus and Rheumatoid Arthritis which are mutually exclusive. Assume that you have 20 cases which are available and that 10 of the cases have been diagnosed by a human expert as Systemic Lupus and the other 10 as Rheumatoid Arthritis. Suppose the number of true and false positives for these two diagnoses, after comparing the expert system diagnoses with the human expert diagnoses, are as given in the table below.

Diagnosis	True Positives	False Positives
Rheumatoid Arthritis	9	5
Systemic Lupus	3	1

SEEK2 first looks at the maximum number of false negatives and false positives which occur. Rheumatoid Arthritis had 1 false negative and 5 false positives and Systemic Lupus had 7 false negatives and 1 false positive. Systemic Lupus will be considered first, since it has 7 false negatives. Suppose that rule r1 concludes Systemic Lupus. Then this rule can be generalized to reduce the number of false negatives. As we mentioned earlier this can be done by either deleting a condition in the rule or increasing the certainty for the rule r1. SEEK2 has a method to decide which one of these to do. After generalizing rule r1 suppose SEEK2 runs the cases again and finds that the number of true positives for both the diagnoses increase from 12 to 15. SEEK2 records this information.

It will next consider the diagnosis Rheumatoid Arthritis and find that the rule for it is r2. It similarly decides how to refine this rule. Assume that, after refinement, the number of true positives go up to 13 for both the diagnoses. Since the refinement of rule r1 has led to a better improvement (in terms of number of true positives) than the refinement of rule r2, it will decide to accept the refinement of the rule r1.

So the basic method used by SEEK2 is to:

1. Consider a set of diagnoses for which rules have to be refined.

2. Find the number of false positives and true positives for each of the diagnoses.

3. Find the diagnosis which has the maximum number of false negatives or false positives.

4. Find the rule corresponding to that diagnosis and determine (based on some method) the refinement which needs to be done to the rule.

5. After refining the rule, determine the improvement in the number of true positives for all the diagnoses together.

6. Repeat steps 4 and 5 for the remaining diagnoses and choose the refinement which leads to the maximum improvement. Carry out that refinement on the rule base.

Though an automated tool like SEEK2 would greatly help, some of the ideas mentioned here can also be used in manually trying to refine a rule base. There are in fact expert system shells which allow one to archive and run test cases on a system in order to determine the performance of the system.

As was seen from this section, cases for a domain provide valuable input which can be used to refine an expert system. This, along with the detection of types of problems described in the previous section, could help greatly improve the performance of an expert system.

9.5 Validating Expert Systems

There are several ways in which expert systems can be validated. Depending on the characteristics of the domain and the constraints the implementor has, one or more of these methods may be chosen.

In certain domains it is fairly straightforward to decide whether the conclusions which the expert system comes up with are correct or not. These domains include domains such as equipment fault diagnosis where you have defective equipment and can diagnose the fault using other methods. Other domains, where there are case histories giving previous cases with experts' conclusions on these cases, can also be validated using this information. In both these types of domains, the expert system can be validated by the expert system developer.

An expert system can be validated by giving the same set of cases to the expert system and to a domain expert. The results of the expert system can then be compared with the domain expert to see if they agree or not. In cases where there is a disagreement, the domain expert could be asked to look at the reasoning which the expert system uses and point out any problems.

In domains where the domain experts normally agree, an expert system can be validated by allowing a number of experts to run the system through different cases and deciding whether the system comes to the same diagnosis as theirs', for the same input. The experts can also be asked to check the reasoning of the system and see if it is correct or not. The experts who are involved in validation should ideally not include any of the domain experts involved in the development of the expert system, to avoid any bias which may creep into the validation.

In domains where the domain experts do not typically agree, the expert system can be validated against the domain expert who was involved in the development of the system. The domain expert can be asked to look at the rules which have been created and see if they correspond to the knowledge he applies in that domain. The domain expert can also be asked to run test cases on the system to see whether it gives the desired results.

Field testing is a common method used for validating expert systems. By field testing we mean that the expert system should be put in real operating environment and its performance observed over a period of time. Based on these observations, one can find out whether the system is performing upto its expectations or not. In some domains, field testing may not be directly possible. In those cases, the system may be tested with real data in a lab. Though this testing will not be as effective as real field testing, it will help one get an indication of the system performance, and point out problems in the system.

Finally one should remember that validation can never be perfect. Even if a combination of the techniques described above are used, there is still a chance that some problems may remain and turn up later. Validation however helps to reduce the number of such problems.

9.6 Conclusion

Verifying and validating expert systems are important tasks in the development of an expert system. These tasks are often more complex in expert systems than in conventional programs. As in conventional programs, the first version of an expert system is almost *never* correct. The system needs to be closely examined, and refined to make it perform better. This task can often be tedious.

In this chapter we have tried to describe some of the most common problems which occur in rule based systems. These are the sort of problems one should look for in expert systems. Some of the high end expert system shells, in fact, provide automated tools which do verification of rule bases and allow validating of an expert system against test cases. Other tools like SEEK2 which come up with refinements for a rule base are likely to be available in the future to help a knowledge engineer refine and improve a knowledge base.

Summary

- Verification, refinement and validation of expert systems are important and need to be continuously done during expert system development.
- Verification deals with determining whether the system's knowledge satisfactorily represents the domain knowledge.
- Refinement deals with improving the performance of the system.
- Validation deals with trying to determine whether the system does what it was intended to and at an adequate level of accuracy.
- There are a number of different types of problems which can occur in rule based systems. Some of these problems can be detected by systematic inspection of a rule base.
- The dependency chart can be used to detect circularity in rules.
- One way of refining a rule base is to run a number of test cases on it. Depending on the performance one can try to determine what rules need to be refined and how.

- The validation technique used for an expert system depends on the characteristics of the domain.
- Some expert system shells provide tools which can help in the verification, refinement and validation of knowledge bases.

Exercises

1. Prevention is obviously better than cure. Can you think of ways to avoid some of the problems which occur in rule based systems?
2. In a backward chaining system, while running test cases for a particular diagnosis, you come up with 4 false positives. What could be the possible reasons for these false positives?
3. On running a set of test cases for another diagnosis you get 3 false positives and 3 false negatives. The rules causing the false positives are in a set S1 and the ones causing the false negatives are in a set S2. Can S1 and S2 be identical?
4. Are there other ways of validating an expert system (which are not given in this chapter) which you can think of? What are they, and for what sort of applications are they applicable?
5. You have developed an expert system for a large insurance company which is supposed to advise people, based on their responses to questions, what policies they are eligible for and what their premium would be. How would you go about validating this expert system?
6. If, instead of an insurance expert system, you had an expert system for determining faults in a nuclear power plant, would your validation strategy change?

10. Advanced Topics

10.1 Introduction

While expert systems have been claimed to be almost the only commercially successful side of Artificial Intelligence, the area continues to present challenging problems to the academic world. There are many issues in expert systems technology which require better solutions. Some of the major issues and the current directions of work in these areas are highlighted in this section. Later sections provide more detail on some of these issues.

10.1.1 Automatic Knowledge Acquisition

Today, expert system development requires a human Knowledge Engineer (KE) to elicit the relevant knowledge from experts. The KE then represents this knowledge in one of the representation formats available in the system being used. The main problem that a KE faces is in getting the expert to articulate the relevant knowledge. Typically experts do not know what they know. Given a case, they can solve it efficiently, but it is difficult for most experts to abstract the methodology and relevant considerations related to a problem. Knowledge Engineering is discussed in more detail in Chapter 8.

Hence researchers are pursuing attempts at automating, as far as possible, the task of a KE. The ideal would be a machine which can talk to the expert directly in natural language (conducting interviews, case analysis etc.) and acquire the relevant knowledge. This today is only a dream, and at least a decade or more away from reality. The main directions being pursued in the area of knowledge acquisition are discussed below.

Intelligent Editors for Knowledge Bases

If the editor used to create the knowledge base, knows the structure and interpretation of objects in the knowledge base, it can help the KE in many ways. These include checking new knowledge for consistency with what is available and pointing out problems like circularity, redundancy, contradiction etc. (See Chapter 9 for details.) These problems otherwise have to be tracked down by hours of debugging, typically after problems are detected during testing.

Systems such as TEIRESIAS [Davis, 1979] go a step further. They maintain internal models of the rules currently in the knowledge base and match new rules against the models. Hence the system can at times come up with suggestions such as "I notice that most rules so far with conclusions about the attribute disease use the attribute suspected_disease in their antecedents. The new rule just added does not. Can I try to add an appropriate condition?". TEIRESIAS also provided debugging support to the KE in interactively tracking down the rule corresponding to a faulty behaviour. In Chapter 9, another system named SEEK2 is described, which provides a method by which rule base refinement may be partially automated.

Generating Rules from Case Histories

In many application domains, records of past cases are available in large numbers, for example, patient records in a hospital. The domain knowledge of experts can be in a way, visualised as an abstraction of a large set of previously encountered cases. We can think of equipping a machine with the power of abstraction and letting it induce the knowledge from a set of solved cases presented to it. The idea is that, just like human beings, the machine can also learn from experience. This is part of the highly active area of research in machine learning. For those interested, [Shavlik and Dietterich, 1990] contains a collection of classic papers covering most areas of machine learning. Other relevant references in this area are [Michalski et al, 1983] and [Michalski et al, 1986].

We are not interested in machines that learn to walk by trial and error, nor in machines that can be taught by interaction. These are still complex research problems. We are concerned about abstracting general principles out of many instances of data. The area of Machine Learning is currently very active with many real world successes demonstrated in specific domains. There are systems, such as RuleMaster, available in the market that can do rule induction using case data. We look at rule induction in more detail later on.

Automatic Rule Refinement

Rule induction methods available today require black-and-white decisions. In other words, every case should have a definite conclusion. In domains where uncertainty or multiple answers are the way of life, these approaches run into trouble. Even the conventional notions of consistency etc. in a rule base are not valid when the rule base contains numeric models of uncertainty (see Chapter 9).

Thus debugging such rule bases is also more difficult, because a faulty conclusion can be explained by a great many causes and isolating the location to a reasonable subset of these causes is difficult without a deep knowledge of the domain. Some research efforts are targetted in this direction. The idea is to collect a number of cases where the expert system has made the same faulty conclusion and to analyse the rule-firings and status of various antecedents in these cases. Then one can statistically narrow down the set of possible causes to a manageable set. The KE can then choose from these possible causes, and correct the problem. In some systems like SEEK2 [Ginsberg, 1988] the system could even identify the precise correction required.

This approach also requires case records along with the expert's decision on each case. When the system decides to apply a refinement to correct a fault, it can verify the effectiveness of the correction by re-running all past cases with the revised rule base to ensure that the fault is rectified and that no new fault has been introduced.

10.1.2 Improving the Knowledge Representation

Most early expert systems used if-then rules with certainty factors for uncertainty handling. Second generation expert systems require more flexibility and power than what can be obtained from this representation. Large expert systems with over 10,000 rules are coming into use. An unordered set of so many rules can be extremely difficult to maintain. Pure rule based systems also have problems in representing certain classes of knowledge in a natural way. Hence there is a lot of interest in using frame like structures along with rules to represent the domain knowledge (see Chapter 4 for a description of frames).

Similarly, uncertainty handling methods, such as those used in MYCIN, have drawbacks as mentioned in Chapter 5. Two other paradigms, Dempster-Shafer theory and Fuzzy Logic, have been explored by ES researchers. Both are formal approaches, avoiding many of the criticisms levelled against Bayesian and certainty factor (CF) based methods. We examine these two mechanisms in detail later on in this chapter. [Shafer and Pearl, 1990] is an excellent survey of various approaches to handling uncertainty in Knowledge Based Systems.

Another aspect of uncertainty handling ignored in most early systems is handling of incompleteness. This requires what is called *Non-Monotonic Reasoning* (NMR). The basic idea is as follows: If you do not know some information you may *assume* something that is generally valid. To take the classic example of flying birds, unless otherwise specified you may assume that any bird can fly. However, later on, you may discover that the bird being considered had its wings clipped in an accident. Then you should be able to withdraw the earlier assumption that the bird can fly. This will also require other conclusions that were drawn based on this assumption to be withdrawn in a similar manner. Systems providing facilities to handle such reasoning are said to be Non-Monotonic.

Various proposals have emerged for a basic framework for NMR. Since *being able to falsify something that was held to be true for sometime*, is not possible in classical logic, NMR essentially takes us out of the realm of standard predicate logic. Reiter's default reasoning [Reiter, 1980], McDermott's non-monotonic logic [McDermott and Doyle, 1980] and McCarthy's circumscription [McCarthy, 1980] are attempts to provide a formal framework for NMR. We will not deal with NMR in detail in this book.

Truth Maintenance System (TMS) [Doyle, 1979] provides an implementation mechanism for non-monotonic reasoning. Every time you derive a conclusion, the system records all information used to derive the conclusion, which essentially forms its *justification*. If one of the current conclusions, say C1, is later shown to be invalid, the system can use the *dependency records* created by it to identify all other conclusions that were derived based on C1 being true. If C1 was the only justification for such a conclusion C2, then C2 will be marked as invalid. This may further lead to other conclusions being invalidated. Some of the high end expert system shells available today (eg. ART, CLIPS etc.) support truth maintenance.

These developments will enhance the power of expert systems in terms of capabilities and efficiency, enabling them to be used for a wider range of problems.

10.1.3 Model Based Reasoning

One of the main application areas for expert systems has been troubleshooting of machinery, especially electronic circuits. A conventional expert systems approach requires a good human expert in the domain. Years of training and experience are required to develop expertise in troubleshooting a particular piece of circuitry. By then, the circuitry may become obsolete. Hence we need an approach that reduces the dependency on a human expert.

One way to do this is to reason using a model of the circuit. Knowing what the components present are, their interconnection patterns and how they interact, it should be possible for the system to explain any faulty behaviour in terms of some faulty components. This is the basic idea of Model Based Reasoning (MBR). There is no clear notion of *expertise* involved in this approach. We discuss this topic in more detail later on in this chapter.

This method has shown good results with electronic circuit fault analysis where the nature of components and interconnections are well-known. However, extending this approach to complex machinery is still a research topic. The main question is how to model a complex equipment.

Having given an overview of the active areas of research in expert systems, we now examine some of the more practical and useful ones more closely.

10.2 Inducing Rules from Cases

A good part of human learning makes use of induction. The main idea is to generalise from a set of examples to infer some abstract rules. Observing that most patients found to have jaundice have yellow eyes leads us to the hypothesis that jaundice causes yellow eyes. This kind of observation helps an expert to improve his problem solving efficiency by formulating this hypothesis in the reverse as: *if yellow eyes are noticed, it is likely that the person has jaundice.*

Consider the table in Figure 10.1 classifying a set of 9 objects as satisfying a set of conditions (YES) or not (NO). In this particular example, the conditions are that the object be small in size and blue or red in colour. We record values for a set of attributes which are possibly relevant in deciding the outcome of the classification.

A few minutes' observation of this table should enable one to formulate a general rule to predict if a given object belongs to this class or not, based on the three attributes shape, colour and size. It would be something like this:

No	Shape	Colour	Size	Yes/No
1.	round	red	small	yes
2.	square	red	small	yes
3.	round	blue	large	no
4.	round	red	large	no
5.	square	blue	small	yes
6.	square	blue	large	no
7.	round	green	small	no
8.	round	green	large	no
9.	square	green	small	no

Figure 10.1: Sample Data for a Decision Tree Induction Example

If size = small and colour = blue **or** red **Then** answer = yes **else** answer = no.

This is in standard expert system rule format. Today's expert system tools expect the knowledge engineer to derive such rules from case records and discussions with human experts, and enter the rules into the system. Considering the difficulties of knowledge acquisition, the question we are interested in is the following: Given a set of case records as in the table in Figure 10.1, is it possible for a system to automatically arrive at rules to deduce the conclusion? This is what the area of rule induction is all about.

One of the most popular classes of work in this direction is from Quinlan's group at the University of Sydney. They have developed a family of rule induction systems called *TDIDT* (Top Down Induction of Decision Trees). Before we describe this work, we will formalise the notion of rule induction.

Consider a collection of objects, say U. These could be concrete objects such as animals or abstract ones such as the weather at a place. Each object is described by a set of attributes and their values. For example a particular instance of weather at Bombay can be described by the attribute-values [temperature:normal, season:summer, cloudy_sky:false, pressure:steady]. Note that the set of possible attributes are pre-defined. The particular value taken by an attribute depends on the object. We also assume, for now, that the values of all attributes are known for every object in U.

Then we have a set of classes into which we want to classify the objects. In our simple example, the classes may be called *yes* and *no*. For a diagnosis system the classes would be the set of faults. We assume the classes to be mutually exclusive and exhaustive. That is, any object of interest should belong to one and only one of the classes.

The class of each object in U is known. The task is to make some rules based on this table so that when a new object (which may or may not belong to the set U) is given, we can predict the class it would belong to.

This knowledge could be represented in two ways. We could make one rule for each class of interest. The class of an object would be the one corresponding to the rule whose conditions are satisfied by the object. Another way would be to construct a decision tree for the whole problem. The intermediate nodes of the tree represent tests on the various attributes, similar to the conditions of a rule. The leaf nodes represent classes. To classify an object, one traverses the tree, starting at the root, taking the branches whose tests are satisfied by the object. The class corresponding to the leaf node reached in the tree is the class of the object. A sample decision tree for the example is shown in Figure 10.2.

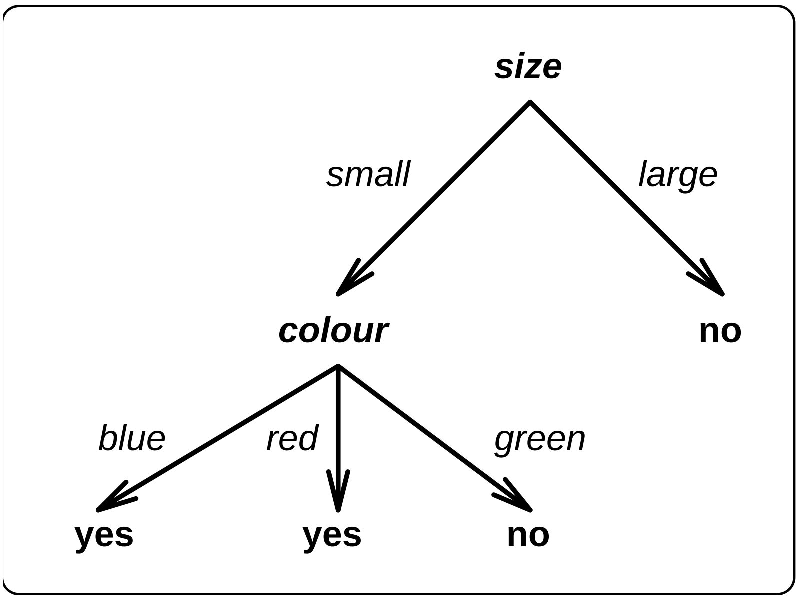

Figure 10.2: Sample Decision Tree for the Data in Figure 10.1

TDIDT, as the name suggests, generates decision trees for classification. Others like Michalski [Michalski, 1983] follow different approaches. In this chapter, we restrict our discussion to the TDIDT family of systems.

The basic idea of constructing a decision tree is simple. To start with, we have the whole class of objects U. We arbitrarily choose an attribute, A1, and consider the possible values that it can have. Assume that it can have one of three values V1, V2 and V3. Based on the assumptions we have made for the members of U, the set U can be split into three subsets - the objects which have value V1 for A1, those with value V2 for A1, etc. Call these subsets C1, C2 and C3. In general, these subsets will not be homogeneous, that is, they will have objects of more than one class in them. For non-homogeneous subsets, we choose another attribute and apply the same process. At each point we sub-divide the set into smaller sets. Finally, we will arrive at subsets such that all members of a subset belong to one class.

Obviously, depending on the order of attributes chosen for testing, we will get different decision trees. For instance, for our example given above, Figure 10.3 shows another correct decision tree. Though all such decision trees are valid and correct, some of them are significantly simpler than others. For example, the tree in Figure 10.2 is simpler than the tree in Figure 10.3. A simpler decision tree is easier and faster to use. Hence we would like our algorithm to generate trees that are as simple as possible. Examining all trees and choosing the best one is, however, not feasible, due to the large number of trees that are possible.

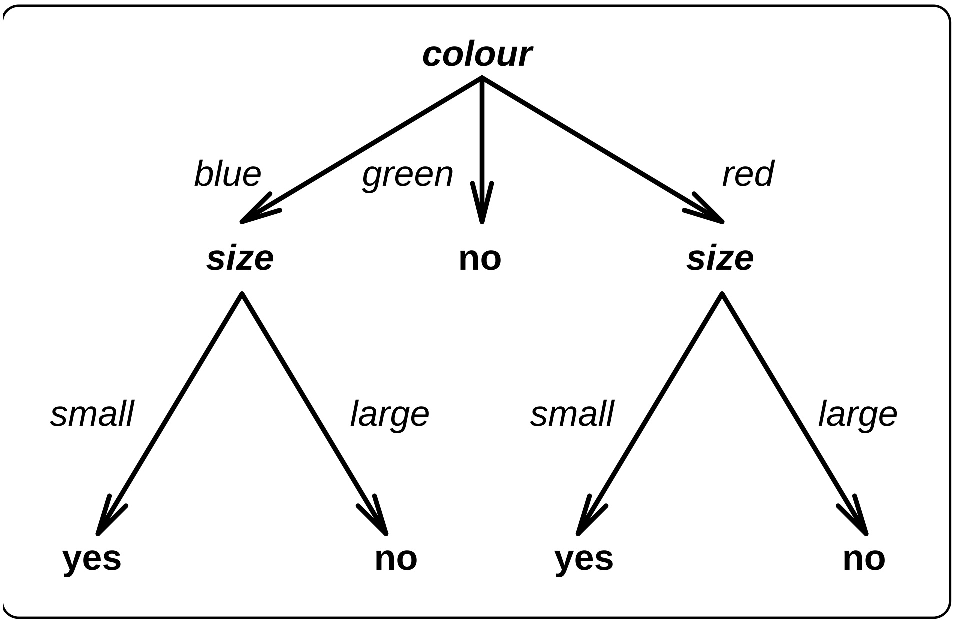

Figure 10.3: Another Decision Tree for the Example given in Figure 10.1

TDIDT uses a variety of statistical techniques as heuristics to generate simple decision trees. One of the early successful TDIDT algorithms was ID3 [Quinlan, 1979] [Quinlan, 1986]. ID3 could handle only attributes with a discrete and finite set of possible values. A later program, ACLS, extended this to handle unrestricted integer values. Another extension, ASSISTANT, provides for real-valued attributes too. It also allows a hierarchy of classes to be used.

We will now look at ID3 in some detail.

10.2.1 The ID3 Algorithm

A small subset W is chosen from the set of objects U. A decision tree is generated for this subset first. The resulting tree is tested against the remaining elements of U. If all of them are correctly classified, the algorithm terminates. Otherwise, a few of the wrongly classified cases are added to W and the process is repeated.

Given a set C of objects (to start with, this will be W), the system computes a heuristic value for each of the attributes. The attribute which gives the maximum value for this parameter is chosen as the next test. If this attribute can have one of R values, the set C gets partitioned into R different subsets (some of them possibly empty). The chosen attribute is removed from consideration and the process repeated for each of the R subsets, excepting those whose members all belong to a single class.

The heuristic parameter chosen is based on information theory. The idea is to find the attribute which has the most discriminatory power. For simplicity we will assume that the number of possible classes is two (for example, *Does the patient have a particular disease or not?*), say P and N.

Consider attribute A. Assume O1, O2, ..., On are the possible values that A can take. The C objects will then get partitioned into n classes C1, C2, ..., Cn corresponding to the values of the attribute A. Let subset Ci contain pi objects of class P and ni objects of class N. Consider a sub-tree of the full decision tree starting at one of the subsets Ci. A parameter called the information associated with this sub-tree is defined as: [1].

$$I(pi, ni) = \frac{-pi}{pi+ni} \log \frac{pi}{pi+ni} - \frac{ni}{pi+ni} \log \frac{ni}{pi+ni}$$

The total information associated with all the sub-trees of the current node (testing on attribute A) is the weighted sum of the information associated with these sub-trees.

$$E(A) = \Sigma i \frac{pi+ni}{p+n} I(pi, ni)$$

where p is the total number of objects of class P present in C and n is the total number of objects of class N.

I(p,n) can be viewed as the information associated with the sub-tree rooted at the current node. The attribute that maximises I(p,n) - E(A), is selected as the next test. This parameter can be viewed as the expected information gain from using attribute A for the next test, which is essentially a measure of the discrimination power of the attribute A.

Example: Consider the data given in Figure 10.1 above. We start with an empty tree (which has no tests chosen yet). We have to choose from shape, colour and size for the test at the root node. Since U has only 9 elements we use the whole set for induction (i.e., W = U).

In all, there are 9 cases, of which 6 are negative and 3 positive. Hence the information associated with the whole tree is:

$$p(3, 6) = \frac{-3}{9} log(\frac{3}{9}) - \frac{6}{9} log(\frac{6}{9}) = 0.918$$

Now we consider the three possible attributes (size, shape and colour) as candidates for the next test and study the information content of the resulting sub-trees.

A = size There are two possible values: *small* and *large*.

1. Value *small*: total 5 items with size=small. Of these positive items = 3, negative items = 2. Hence pi=3, ni=2.
   $$p(3, 2) = \frac{-3}{5} log(\frac{3}{5}) - \frac{2}{5} log(\frac{2}{5}) = 0.97$$
2. Value *large*: total 4 items. pi=0, ni=4. When either pi or ni becomes zero, it means there is no further information required to identify that subclass and hence the p for this is taken as zero.

Hence, $E(A = size) = \frac{5}{9} * 0.97 + \frac{4}{9} * 0.0 = 0.54$

A = colour There are three possible values: *red, blue* and *green*.

[1] Those interested in finding out the mathematical details of the parameter may refer to [Quinlan, 1986].

1. Value *red*: total 3 items, 2 positive and 1 negative
   $p(2,1) = \frac{-2}{3}log(\frac{2}{3}) - \frac{1}{3}log(\frac{1}{3}) = 0.918$
2. Value *blue*: total 3 items, 1 positive and 2 negative
   $p(2,1) = p(1,2) = 0.918$
3. Value *green*: total 3 items, 0 positive and 3 negative
   $p(0,3) = 0$

Hence,
$E(A = colour) = \frac{3}{9} * 0.918 + \frac{3}{9} * 0.918 + \frac{3}{9} * 0.0 = 0.612$

A = shape There are 2 possible values: *round* and *square*.

1. Value *round*: total 5 items, 1 positive and 4 negative
   $p(1,4) = \frac{-1}{5}log(\frac{1}{5}) - \frac{4}{5}log(\frac{4}{5}) = 0.72$
2. Value *square*: total 4 items, 2 positive and 2 negative.
   $p(2,2) = \frac{-2}{4}log(\frac{2}{4}) - \frac{2}{4}log(\frac{2}{4}) = 1.0$

Hence, $E(A = shape) = \frac{5}{9} * 0.72 + \frac{4}{9} * 1.0 = 0.844$

The information gain from the attributes can be tabulated as follows:

size	(0.918 - 0.54) = 0.378
colour	(0.918 - 0.612) = 0.306
shape	(0.918 - 0.844) = 0.074

It is clear that the most information gain will be got by choosing the size attribute as the current attribute. This leads to 2 subclasses: one with size = *small* and the other with size = *large*. The *small* subclass contains positive and negative elements and hence needs to be further refined in this fashion. The *large* subclass contains only negative examples and hence the resulting node can be labelled as negative (NO).

It can be shown that in refining the size = *small* subclass, the attribute chosen will be colour. Thus in this example, we get the simple decision tree shown in Figure 10.2.

10.2.2 Prospects of Rule Induction

Expert-Ease, Ex-TRAN and RuleMaster are commercial systems which use ACLS. These tools have been used successfully for developing expert systems for various applications.

The algorithm used in ID3 has also been found to work fairly well in cases where the data is noisy (that is, where some of the objects in U are wrongly classified) and incomplete (where some objects are not completely described). This was observed in an experiment at the Garvan Institute of Medical Research, Sydney, where this approach was used to generate an expert system for diagnosis of thyroid patients. About 3000 cases were used as the training set. The system achieved over 99% correctness in diagnosis.

The AQ11 program for inducing decision rules, developed by Michalski et al [Michalski, 1983], has been found to generate rules that performed better than the traditional knowledge engineering approach. In a case where this program was used to identify diseases in soybean plants, Barry Jacobson, a soybean expert, coded rules in the conventional way. There were 19 diseases to be considered. There were 35 attributes used to describe a case. Using 307 training cases, the AQ11 program generated a rule base. This was tested on another 376 cases, along with the expert-generated rules. The expert-rules could achieve a maximum of 93% correctness after a lot of hand-tuning, whereas the induced rules easily obtained a correctness of 99%.

Thus, the rule-induction methodology has been proven to be ready for real-life application and shows great promise at least in select areas where past case records are available. One problem that held back large scale application of this approach was the fact that the induced rules may not make sense to the domain experts, making it difficult for them to make any refinements later on. The work by Shapiro on structured induction attempts to address this problem [Shapiro, 1987].

Also, extending the methodology to handle domain uncertainty (where a conclusion cannot be drawn with full certainty most of the time) is still not addressed by this mechanism. The current state of rule induction may be best summarised by the comment: "*they [the rule induction methods] appear unlikely to capture much of the subtlety or complexity of human decision making and will probably be limited to use in data rich and theory poor domains*" [Grimson and Patil, 1987].

10.3 The Dempster-Shafer Theory

The Dempster-Shafer theory is a mathematical approach developed by Arthur Dempster in 1960 and subsequently extended by Glenn Shafer. The decision problem is viewed as choosing one element out of a set of hypotheses. This set of hypotheses, which we will denote by F, is called the Frame of Discernment. The Frame of Discernment has to be a mutually exclusive and exhaustive set. In other words, for any given case, there must be one element in this set which is the answer and there must not be more than one element which is the answer. This is a must for the Dempster-Shafer (DS) approach to work and is one of the major problems in implementing the approach.

Consider a medical example (adapted from [Gordon and Shortliffe, 1984]) in the domain of cholestatic jaundice. Cholestatic jaundice is essentially elevated bilirubin levels in the patient's body. Bilirubin is a pigment produced by the liver. The reason for this elevated bilirubin levels could be due to a problem within the liver (intrahepatic) or a blockage of the bile ducts (extrahepatic). We will consider four causes for cholestatic jaundice, namely, Hepatitis (hep) and Cirrhosis (cirr) belonging to the intrahepatic class and Gallstones (gall) and Pancreatic cancer (pan) belonging to the extrahepatic class. The hypotheses space can be represented as a hierarchy as shown in Figure 10.4. One of the plus points of the DS approach to uncertainty is its ability to make use of such a hierarchy in the domain hypotheses.

A piece of evidence is viewed as supporting or rejecting some subset of F

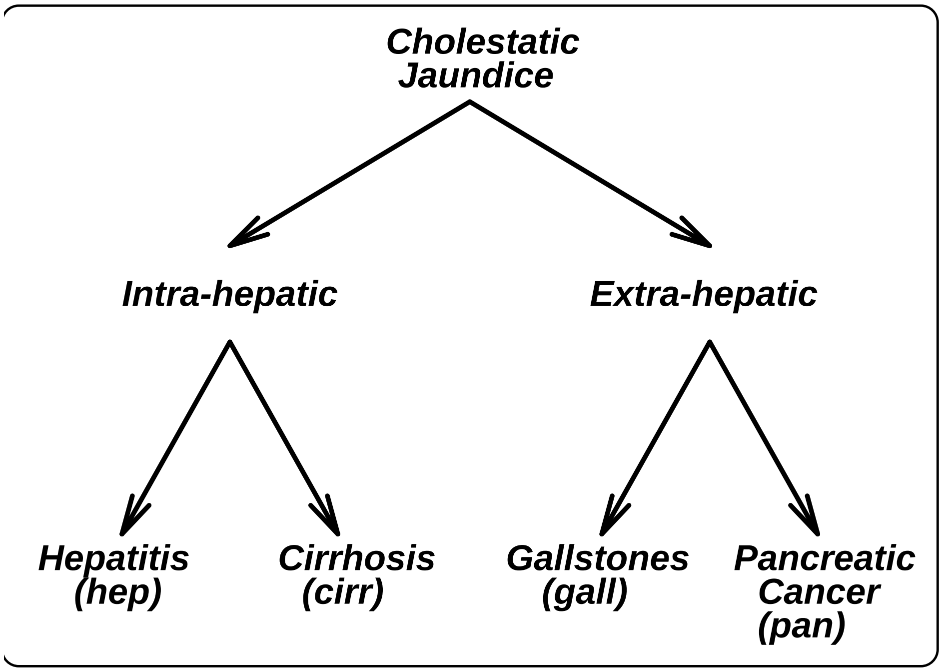

Figure 10.4: The Hypotheses Hierarchy for Cholestatic Jaundice

with varying degrees of belief. With each piece of evidence, we attach a *basic probability assignment* (BPA) function. When that evidence holds, this function assigns a belief to each subset of the frame of discernment F, such that the sum of beliefs assigned to all the various subsets of F equals 1.0. Unlike in the MYCIN-type approach, the support does not always have to be for one of the leaf nodes in the hierarchy; it could be for one of the higher level nodes as well (i.e., support a set of hypotheses). This allows the system to come up with a diagnosis such as "I am sure the problem is intrahepatic, but I do not have evidence to decide if it is *hepatitis* or *cirrhosis*".

The BPA function, m1, for such cases, where an evidence supports a subset S of F with a belief P, will be:

m1(S) = P, m1(F) = 1−P and m1(X) = 0 for all other subsets X of F.

When a piece of evidence is to reject a subset S of F with a belief P, the BPA function m2, will be:

m2(S') = P, m2(F) = 1-P and m2(X) = 0 for all other subsets X of F.

where S' denotes the complement of S with respect to F.

That is, evidence against a hypothesis (or a set of hypotheses) is treated as evidence for its complement. Note that the frame of discernment F must be mutually exclusive and exhaustive.

Consider a piece of evidence supporting *gall* with, say, a probability of 0.6. The BPA function for this can be written as m({gall}) = 0.6, m(F) = 0.4 and m(X) = 0.0 for any other subset X of F.

Suppose another piece of evidence supports extrahepatic with some probability 0.8. The BPA for this case will be m({gall,pan}) = 0.8, m(F) = 0.2

and m(X) = 0.0 for any other subset X of F.

10.3.1 Combining Evidences

As in the certainty factor approach, DS also requires a scheme to combine the evidences coming in from various sources into a single belief figure for each hypothesis. However, unlike in the CF approach, we have an additional complication arising from the hierarchical structure. Consider the hierarchy given in Figure 10.4. Any evidence for a node lower down in the hierarchy is also evidence for any node lying in the path from that node to the root.

Hence, the evidence combination in DS theory, proceeds in two steps. First all available BPAs arising from the various pieces of evidence are combined to a single equivalent BPA. Another formula is used to obtain the net belief for any given node in the hierarchy based on the combined single BPA. This latter formula has to take care of the beliefs assigned to nodes lower down.

For any given subset S of F,

$M(S) = \sum(m1(X) * m2(Y))$
for all X and Y such that $X \cap Y = S$,
where $\cap$ denotes set intersection and $\sum$ denotes summation.

It can be seen that M qualifies to be a BPA since the sum of the beliefs assigned by M to all subsets of F will be 1. Note that in the above formula, any given X - Y combination will contribute to exactly one subset of F and all X - Y combinations will be considered. Thus $\sum M(S)$ over all S will be $\sum(m1(X) * m2(Y))$ for all possible combinations of X and Y. Thus

$\sum M(S) = \sum(m1(X) * m2(Y)) = \sum m1(X) * \sum m2(Y) = 1$
(since m1 and m2 are BPAs by definition and X and Y span
all possible subsets of F)

Let us consider an example based on Figure 10.4. Consider the two BPAs m1 and m2 mentioned earlier. The two BPAs are combined to obtain a single equivalent BPA, say M. See Figure 10.5.

For convenience, we will enumerate the subsets X (for m1) on the horizontal axis and the subsets Y (for m2) on the vertical axis. Each point on the intersection of a vertical line and horizontal line gives the intersection of the respective sets on the two axes.

m1 m2	{pan,cirr} 0.5	F 0.5
{cirr} 0.8	{cirr} 0.4	{cirr} 0.4
F 0.2	{cirr,pan} 0.1	F 0.1

Figure 10.5: Example for Combining BPAs – a Simple Case

Note that we have shown only those subsets which have a non-zero BPA assigned. The other rows and columns which are not shown will give zero BPA for the intersection sets and hence will not affect the final result.

Thus the combined BPA, M from m1 and m2 will be:

M({cirr}) = 0.4 + 0.4 = 0.8

note that both the BPAs support {cirr}

M({cirr,pan}) = 0.1

M(F) = 0.1

M(X) for all other subsets = 0.0

It can be seen that $\sum M(S)$ over all subsets S of F is equal to 1.0 (that is, 0.8 + 0.1 + 0.1)

Consider a new evidence with the BPA function m3 as given below:

m3({pan}) = 0.6

m3(F) = 0.4

The table for obtaining the new combined BPA will be as shown in Figure 10.6.

M m3	{cirr} 0.8	{cirr, pan} 0.1	F 0.1
{pan} 0.6	{} 0.48	{pan} 0.06	{pan} 0.06
F 0.4	{cirr} 0.32	{cirr,pan} 0.04	F 0.04

Figure 10.6: Combining BPAs: A more Complex Case

Now there is a complication. The null set (which in this case implies that the patient has no disease) should never be true, since our assumption is that the set F is exhaustive. However, it has now got a non-zero belief (0.48) in the resulting BPA. The DS approach circumvents this problem by removing the belief assigned to null sets and, normalising all other beliefs accordingly.

The net belief assigned to the null set is 0.48. Set that belief to 0.0. The remaining belief is 0.52. We will treat this as the net belief and normalise all the other beliefs. To normalise the other beliefs divide every one of them by this new net belief.

The new combined BPA function M' will be

M'(pan) = (0.06 + 0.06)/0.52 = 0.23
M'(cirr) = 0.32/0.52 = 0.61
M'(cirr,pan) = 0.04/0.52 = 0.08
M'(F) = 0.04/0.52 = 0.08

Thus as the evidence accumulates, the belief on each node gets revised.

10.3.2 Belief Intervals

The net belief in a compound hypothesis like extrahepatic is not just the belief for {pan,gall}. It should also include the belief associated with {pan} and {gall}. Thus the net belief assigned to a node is calculated as the sum of beliefs assigned to all nodes in the hierarchy rooted at the required node. In other words, it is the sum of the belief of the node and the beliefs of all its descendants.

Consider the total belief assigned to a subset of F, say {pan,gall}. This will include the belief assigned to the node marked {pan,gall} as well as the beliefs assigned to the nodes marked {pan} and {gall}. Thus you can compute beliefs on any group of hypotheses as well. We will denote the net belief assigned to a set of hypotheses s, by Bel(s). This function is defined as:

$Bel(s) = \sum M(x)$ with x ranging over all subsets of s.

Consider the value of Bel(not s). Based on the assumptions on the Frame of Discernment, Bel(not s) is Bel of the complement of s. In our example s is {pan,gall}. Therefore the complement is {cirr,hep}. Bel(not s), hence, amounts to Bel({cirr,hep}). By definition, this quantity is the sum of M({hep,cirr}), M({hep}) and M({cirr}). Note that by the definition of a BPA,

$\sum M(x)$, where x ranges over all subsets of F = 1.0

The subsets of F we have considered for Bel(s) and Bel(not s) form only a part of all the subsets of F. Hence,

Bel(s) + Bel(not s) <= 1.0

This shows that you can have a belief that is assigned to neither s nor its negation. Contrast this with the classical probability model where P(s) + P(not s) = 1.0 always.

The DS approach defines another useful parameter called Belief Interval which characterises the unassigned belief and hence the uncertainty associated with the hypothesis. The Belief Interval of a set `s` is defined as `[Bel(s),   1 - Bel(not s)]`.

The maximum length of this interval is 1.0 and the minimum length is 0. The maximum is attained when you have belief neither in s nor in its negation, that is, you do not believe s nor do you disbelieve s. The minimum is attained when the length of the interval is zero. That is, there is no uncertainty due to lack of information. All the belief has been assigned to either s or `not` s. Note that even when the belief interval is of zero length, you can have equal belief in s and `not` s, and thus not be able to conclude anything decisively.

The length of the belief interval gives a measure of the uncertainty regarding the status of s. The maximum of 1.0 indicates that no evidence has been gathered about s. No belief has been assigned to s or its complement. The minimum of 0.0 shows you have used all the evidence available. However, it may happen that the evidence is equally divided with Bel(s) = 0.5 and Bel(not s) = 0.5. This shows that it is impossible to decide if s is true. Whereas if the interval is greater than 0, there is more evidence that can differentiate between the two. Pictorially, the interval is as shown as in Figure 10.7.

```
        Bel(s)                        1-Bel(not s)
0........^...............................^.............1
<--------><------------------------------><------------>
belief in                                  disbelief in
set s                                         set s
```

Figure 10.7: Belief Intervals

As more and more evidence is gathered, the interval shrinks continuously. Evidence for s shrinks to the right and evidence against s shrinks to the left.

10.3.3 Merits and Demerits of DS Theory

The DS theory has some nice properties from the point of view of expert systems. First, it does not require all evidence to be assigned to individual hypotheses. You can assign beliefs to groups of hypotheses. The DS framework provides a formula to combine these beliefs. This allows the user to make use of a domain hierarchy of hypotheses in an intuitive way. Secondly, partial evidence for a hypothesis is not construed as providing partial evidence for the negation of the hypothesis.

One of the main merits of the DS approach is that there is a clear distinction between ignorance and equal certainty. If you have equal evidence for and against a subset, s, of F, then the set s and its complement with respect to s will have equal measure of belief. Ignorance about a hypothesis, where you have no information about a hypothesis is represented with a belief of zero.

However, there have hardly been any real implementations of the DS theory for any expert system applications. The main reason for this is that, in the general case, the evidence combination formula has an exponential time complexity. There have been attempts to provide optimisations to the formula by identifying special cases [Gordon and Shortliffe, 1985]. But most of these introduce assumptions that go against the basic ideas of the approach. One such simplification was to assume that the different pieces of evidence for a hypothesis are independent. This kind of simplification does not get us very far, since the invalidity and unacceptability of this assumption constituted the major objection to the probability based approaches.

Another problem with the approach is the requirement of an exclusive and exhaustive set of hypotheses. Both these characteristics are difficult to satisfy in many domains. The exclusivity requires that no case will require more than one element from the set to be selected. For a medical domain, this translates to a *one disease at a time* requirement. In order to allow a patient to have more than one disease at a time, we have to formulate all subsets of the set of diseases as the frame of discernment. If you consider a set of 10 diseases, the set of subsets will have 2^{10} elements. Thus we have to deal with very large sets even for a reasonably small set of hypotheses.

10.4 Fuzzy Logic Approach

Fuzzy set theory was proposed by Zadeh and overcomes some of the problems of the DS model and the CF approach. This formalism is essentially an extension of the DS model to handle *fuzzy* evidences. Probability theory captures the randomness associated with an event. Fuzzy set theory, which is essentially a possibility theory, addresses the notion of vagueness or lack of precision in information. This allows us to capture common sense notions such as tall, warm, big, very big etc. which are imprecise [Zadeh, 1988].

Recently, there has been considerable interest in applying this theory to handle uncertainty in expert system applications. Various other approaches including the MYCIN approach have used some ideas from fuzzy set theory in their *ad hoc* methods for handling uncertainty.

In conventional set theory, set membership is defined in a straightforward way. A query about set membership is always answered with either "yes" or "no". Now consider a query such as "Is Gita tall?". If Gita is above (say) 5'5" we may answer "yes" and if she is 4'0", the answer may be "no". But what if she is in between? In fuzzy set theory, set membership is a function that returns a number between 0.0 and 1.0 indicating the degree of membership. Thus we can define "tallness" as a function of height:

Height	tall?
5'5" and above	1.0
5'3" and above	0.8
5'1" and above	0.6
4'8" and above	0.4
4'5" and above	0.2
less than 4'5"	0.0

Now, given the height of Gita, we can answer if she is tall, with a numeric value.

How do we handle negation? Suppose the statement is "Gita is not tall". What would be the certainty to be attached to it? Fuzzy set theory defines negation as the complement of the assertion. Thus the table for not tall derived from the above table is:

Height	not tall?
5'5" and above	0.0
5'3" and above	0.2
5'1" and above	0.4
4'8" and above	0.6
4'5" and above	0.8
less than 4'5"	1.0

The idea here is that the membership in not tall decreases, as the membership in tall increases. Note that when something is certainly a member of one set (membership 1.0), it is certainly not a member of the negated set (membership 0.0). This agrees with the notion of conventional negation and set membership. Thus, using the appropriate membership function, we can obtain a belief value for any atomic condition (a simple condition or its negation).

In set theory, the intersection of two sets consists of those elements that occur in both sets. In fuzzy set theory, since membership can be partial we need to modify this definition. The value for the membership of any element in the intersection of two sets is defined as the minimum of the membership values of the element in both the sets. Similarly the value for the union is the maximum of the membership values in either set.

The concepts of intersection and union are used in defining possibility values for conjunction and disjunction of conditions, respectively. Thus for a conjunction of antecedents, we take the minimum of the values of both antecedents. Note that this is the same as what MYCIN uses in terms of CFs.

Typical rules from human experts are of the following type:

> **If** pressure is *high*
> **and** temperature is *low*
> **then** open the valve *slightly*.

The italicised words indicate words that are imprecise in these rules. For example, if the temperature is 32 degrees Celsius, is the rule applicable? The major advantage of fuzzy logic based systems is the ability to represent rules like this directly and to reason with them. Most other paradigms require more precise rules. The effect is that the human expert may not feel at home with the rule in the precise form.

This description should give the reader a feel for the basic ideas used by fuzzy set theory. The actual methods and applications require a significant amount of mathematical theory. We will not discuss those here. [Negoita, 1985] discusses some of these issues in detail. It may also be noted that there are hardly any expert systems today which actually use fuzzy logic for uncertainty reasoning.

10.5 Model Based Reasoning

Consider any troubleshooting system. The basic information and knowledge for troubleshooting comes from an understanding of the structure and function of the device. Where the structure and function is not completely understood, for example in the case of the human body, people may use empirical results also. But for man-made devices this information is normally available. With experience, the experts in the domain, may be able to supplement this knowledge with shallow direct association rules from symptoms to certain classes of faults. But when a problem not encountered before comes up, the basic knowledge is used for diagnosis. The question for expert system builders is: can we get machines to do diagnosis using basic knowledge, instead of waiting for shallow rules to be discovered and added to the system? This question leads us to the area of Model Based Reasoning (MBR).

10.5.1 Motivation

The Model Based Reasoning (MBR) approach has many advantages over the conventional method of building expert systems. One need not wait for an expert to be available. Typically, by the time a person becomes an expert in troubleshooting a particular piece of equipment, the equipment may become obsolete. Also in a competitive world ideally your troubleshooting expert system should be in place when the product is announced. This can be achieved if all that is required for development of a troubleshooting system is the structural and functional description of the equipment.

Moreover, the development of such systems becomes more methodical since there is no real knowledge engineering involved. The inputs are well-known ahead of time. Also, since the system's diagnosis is based on a model of the device, the structure of the device will be visible to the user and the system is likely to provide more intelligible explanations to the user (see Chapter 6 also).

The idea of MBR has a wider applicability than troubleshooting of equipment. Under the name *qualitative physics/reasoning*, there has been an attempt to address the issue of common sense reasoning. The lack of "common sense" on the part of the machine has been identified as a major bottleneck in extending expert systems to handle more real-world problems. We often use common sense knowledge such as: a glass at the edge of a table may fall down and break, a ball thrown in the air will eventually come down etc. It is hard to view these as a bunch of shallow rules for all such situations. It is equally unlikely that we all use precise mathematical equations about properties of matter and laws of physics to arrive at such conclusions. Hence the hypothesis is that we are able to use a deep qualitative model of various objects and physical laws for our day-to-day reasoning tasks. Therefore, the field of MBR could, perhaps, address the problem of brittleness in current expert systems. For those interested in Qualitative Reasoning, Volume 24 of the Artificial Intelligence journal (December 1984) is a special volume on qualitative reasoning about physical systems.

In conventional rule based systems, the causality relations in the domain are embedded in flat rules, among other types of knowledge [Clancey, 1985]. This makes it difficult for users to understand their behaviour. In MBR based systems, the reasoning explicitly makes use of the structure and function of the domain and hence provides scope for more intelligible explanations.

10.5.2 Basic Framework of MBR

The basic framework of model based reasoning is shown in Figure 10.8.

In general, if the model of the device matches the actual device, there should be no discrepancy between the predicted behaviour based on the model and the observed behaviour of the actual device. However, when there is a discrepancy, it could be due to two reasons: either the model is faulty (that is, it is not a faithful representation of the device) or the device is faulty. Assuming we have a correct model of the device, we can use the model and the difference in faulty and correct behaviours to find out what fault in the device could have lead to the observed faulty behaviour.

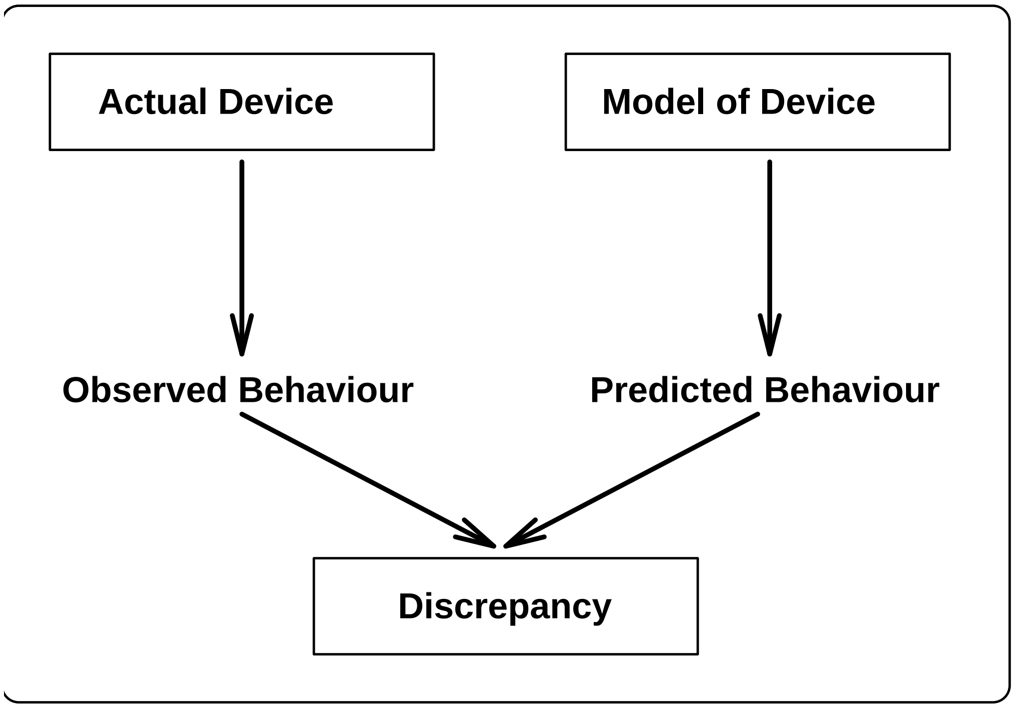

Figure 10.8: Basic Framework of MBR

As an example, consider a simple door latch depicted in Figure 10.9. The functioning of the latch should be clear from the picture. As the door is pushed in from the left side in the figure, the locking arm moves up compressing the spring. When the door is fully in, the arm falls back into the slot in the wedge of the door. To open the latch, you press the knob which pulls up the locking arm with the lever arm acting as a lever. Now you can pull the door out. We can describe the behaviour of this device as follows (omitting many details for simplicity):

Rule 1: knob_down *implies* lever_arm_deflected
Rule 2: lever_arm_deflected *implies* spring_compressed
Rule 3: spring_compressed *equals* locking_arm_up
Rule 4: door_push *implies* locking_arm_up
Rule 5: locking_arm_up *equals* door_can_move
Rule 6: knob_press *implies* knob_down

The word *implies* means that if the left-hand-side is true then the right-hand-side should be true, but not necessarily vice versa. The word *equals* is to indicate this implication holding in both ways (i.e., A equals B is the same as A implies B and B implies A).

For a MBR system based on this, we would mark each of the behaviour rules above with faults which could violate the behaviour. For example, the rule 1 could fail if the lever arm is broken.

Given a case where the user pressed the knob, but could not open the door, we have knob_press and *not* door_can_move.

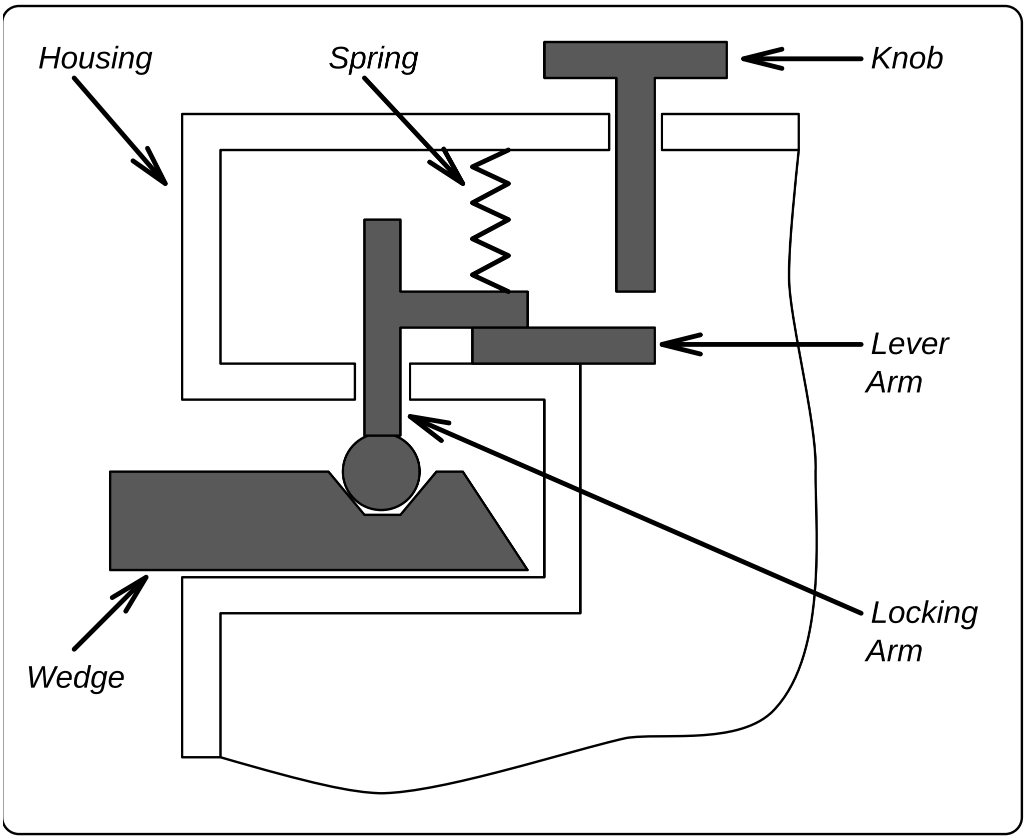

Figure 10.9: A Door Latch

If the latch functions properly, the behaviour rules would deduce the following:

knob_down (from rule 6),
lever_arm_deflected (from rule 1),
spring_compressed (from rule 2),
locking_arm_up (from rule 3) and
door_can_move (from rule 5).

The last inference leads to an inconsistency with the observation that the door cannot be opened (*not* door_can_move).

Hence at least one of the five rules 1,2,3,5 or 6 is not true in the actual device; that is, the component associated with that rule does not obey the rule. Retracting any one of these rules can make the system consistent with the given observation. In other words, a failure in any of the components associated with any of these rules can explain the behaviour observed.

Further observations, like this, can narrow down the possibilities to say, for example, that rule 1 is false. This could mean that the lever arm is broken.

Given a set of axioms describing the behaviour of the device and a set of observations indicating a faulty behaviour, the system would obtain a contradiction if all the axioms are assumed to be true. The MBR methodology first identifies the axioms involved in deriving the contradiction. One of these axioms must be wrong in the actual device. In other words, the component described by one of these axioms is not behaving properly.

If there are more than one suspect axioms thus identified, we require more observations to isolate the fault. Once again, the model we have can help us to suggest tests that are meaningful to narrow down the hypotheses set. If a second set of observations, analysed as above, indicates that one of the rules 2 and 5, is suspect, we can consider rule 2 to be the most likely suspect, since it is able to explain both the set of observations.

We can also conduct tests to check that the axioms are correct. This helps to eliminate those axioms from consideration and thus narrow down the hypotheses set.

The system can use failure probabilities of the various components, where available, to focus on the component most likely to fail, and look for tests to confirm or reject this possibility.

For devices where the fault models of components are known, that is, the way a device would behave under a given fault is known, the fault model can be represented as behaviour rules. In such cases, the system can use such faulty rules (instead of the actual rules for suspected components) to try to derive the observed responses. This provides a way of verifying faults.

This basically is the essence of diagnosis using model based approaches. In the next subsection, we try to define this framework in a slightly more rigorous way.

10.5.3 Formalising the Notion of MBR *

In this section, we attempt to formalise the notion of diagnosis using a device model. We will be using concepts of first order logic discussed in Chapter 3 of this book. The material in this section is primarily based on [Reiter, 1987].

We describe a device by a two-part structure: COMP, a finite set of constant symbols denoting the components in the device and DD (Device Description) which will be a set of first order logic sentences describing the behaviour of the components. For every component, c, in the set COMP, we define a predicate faulty(c) to indicate that the component c is faulty.

In the door latch example discussed earlier,

COMP = {door, locking_arm, lever_arm, spring, knob}

DD is essentially based on the set of rules listed in the previous subsection. But here, we explicitly attach the components required to be working for the rule to be obeyed. Thus the rules in the previous subsection can be written as

Rule 1: not faulty(lever_arm) and knob_down
    implies lever_arm_deflected

Rule 6: not faulty(knob) and knob_pressed implies knob_down

These rules form the device description DD.

We also have a set of observations OBS about the device. These can also be viewed as first order sentences, giving the state of various components of

* This section may be omitted during a first reading.

the device and also the values of observable parameters. For example, an observation could be {knob_press, not door_can_move}.

Consider the set of formulae:

$$DD \cup OBS \cup \{NOTfaulty(c) \mid c \in COMP\}$$

The last set in this formula adds $faulty(c)$ for every component c in the device - i.e., assuming the device is functioning properly. If this formula is consistent, the observation agrees with the model prediction and hence the device is behaving as per expectation. We assume that the device behaviour, DD and observation, OBS are consistent sets by themselves. [2]

Now consider a case where the above set is inconsistent. That is if you assume all components are functioning properly, the observation conflicts with the predictions based on the device model. Since we assume OBS and DD are consistent, the only way to remove the inconsistency is to assume some of the components are faulty. Our task in diagnosis is to choose a *minimal* set A so that

$$DD \cup OBS \cup \{NOTfaulty(c) \mid c \in COMP - A\} \cup \{faulty(c) \mid c \in A\}$$

is consistent. In other words, if we assume all components in the set A are faulty and every other component is working fine, then the set $DD \cup OBS$ should be consistent. Such a set A is called a *diagnosis*.

We can simplify the formulation a bit further. It can be proved that if A is a diagnosis, then for every component c in A,

$$DD \cup OBS \cup \{NOTfaulty(c_i) \mid c_i \in COMP - A\} \Rightarrow faulty(c)$$

That is, the faulty components are implied (can be determined) by the normal components of the device. Hence in the formulation 1, given above, we can drop the last disjunction.

However, if there are N components in the device, there are about N! [3] potential diagnoses. Checking each of these for consistency using the above formulation will be computationally intractable.

The solution is essentially to use the observations about the device to identify a *conflict set*, by mechanisms such as constraint propagation. The device model and the observations are used to identify minimal subsets of components which cannot be all normally functioning at the same time. If such conflict sets are available, we can define the notion of *hitting sets*. H is a hitting set for a set of conflict sets, if H has a non-empty intersection with each conflict set. That is H should contain at least one member from each of the conflict sets.

It can be seen that any diagnosis we arrive at for a given observation has to be a hitting set. Thus instead of exploring all possible subsets of COMP for a diagnosis, we can restrict ourselves to the possible hitting sets of available

[2] Relaxing this restriction would mean that we should also be aware of the faulty model being used for diagnosis.

[3] N! is the factorial of N, defined as $1 \times 2 \times 3 \times \ldots N$.

conflict sets. Once again, the notion of minimality can be used here and only minimal hitting sets need to be explored. This reduces the number of possible options significantly.

[Reiter, 1987] also gives a systematic procedure for efficiently exploring the set of hitting sets. We will not discuss that here.

10.5.4 Current State of MBR

One important advantage of MBR is that a model of all possible faults in the device is not required to perform diagnosis. In practice, it is not necessary or even practical to know the exact faults of a device. Often it is enough to know that a component is faulty, so that you can just replace the component.

The main difficulty in using MBR in practical systems is in having an appropriate way of modelling complicated devices. What level of detail should the model represent? How can temporal sequences of events in machines be represented, especially in machines involving feedback? How can mechanical interactions between components be captured? All these are questions which still require answers before the methodology can be applied more widely. MBR is currently a very active area of research addressing these types of problems.

As of now, there is one class of devices, namely digital circuits, where these questions are easily answered. Most of the work in MBR so far has been in this area, for example systems like DEDALE [Dague et al, 1987], SOPHIE [Brown et al, 1982] and DART [Genesereth, 1984]. There are also attempts to apply these techniques to other domains of troubleshooting [Whitehead and Roach, 1990].

[Davis and Hamscher, 1988] is a good survey of Model Based Reasoning for troubleshooting systems.

Summary

- Rule induction methods (eg ID3, AQ11 etc.) using a large collection of pre-classified cases are proving to be successful for automating knowledge acquisition in some areas. Systems based on ID3 use statistical methods to optimise the resulting set of rules.

- Dempster-Shafer theory offers a mathematically sound mechanism for uncertainty handling, overcoming many of the objections to CF-based approaches. Some of the plus points of the DS theory are that it allows hierarchical hypotheses space, has a notion of belief intervals and can distinguish between ignorance and conflict. However, the exponential complexity of its evidence-combination mechanism and the requirement of an exhaustive and mutually exclusive hypotheses space causes implementation problems.

- Fuzzy logic offers a good mechanism for representing vagueness in the domain knowledge, making the resulting "fuzzy" rules more intuitive to human experts.

- Model based reasoning can make expert systems more robust by providing them with the ability to do deep reasoning. This allows the system to react sensibly to new faults and provide better explanations. The systematicity of the approach and the absence of the knowledge engineering bottleneck makes MBR systems easier to develop. The basic difficulty is in modelling the device to the desired level of granularity and capturing all the relevant component behaviour. The approach has been shown to work well in troubleshooting digital circuits where the component behaviours are well known. Research is in progress to extend the methodology to other classes of devices.
- Non-monotonic reasoning, truth maintenance systems, automatic rule base refinement and rule models are areas of research interest which are relevant to expert systems.

Exercises

1. In a medical research centre, doctors are trying to identify the symptoms of a new disease they have noticed. From hospital records they obtained information about a few patients, some known to have the disease [POS] and others known *not* to have the disease [NEG]. Assume that the only symptoms relevant to this new disease are sympt1, sympt2 and sympt3 with possible values for sympt1 being high, medium and low, and for sympt2 and sympt3 being yes and no. From the table given below, generate a decision tree to find out if a given person falls in the POS or NEG category. Then apply the ID3 algorithm and generate a tree. How does your tree compare with the ID3 generated tree?

Seq No	Sympt1	Sympt2	Sympt3	Class
1	high	no	yes	POS
2	high	no	no	NEG
3	high	yes	yes	NEG
4	high	yes	yes	NEG
5	medium	no	yes	NEG
6	medium	yes	yes	POS
7	medium	yes	no	POS
8	low	yes	yes	NEG
9	low	no	no	NEG
10	low	no	yes	NEG

2. Consider the hypotheses hierarchy for jaundice as given in Figure 10.4. Assume a belief of zero for every node except the root node. For the root node the belief is 1.0. Update the beliefs on the tree for the following sequence of evidences and note the behaviour of the belief intervals on the various nodes in the tree. (Note that the calculations may introduce subsets other than those shown in the tree, for example, {gall,cirr}).

   (a) An evidence supporting intra-hepatic class with a probability of 0.6

(b) An evidence for cirrhosis with a probability of 0.7

(c) An evidence against gall stones with a probability of 0.4

3. Write an algorithm to do model based diagnosis for any device based on the simplified representation used in this chapter. The states of various components are represented by statements which are either true or false (eg. locking_arm_up). The interactions between the components are represented using propositional logic connectives such as *and, or, not, implies* and *equals*. The observations may be assumed to be in the same format used for the model representation — a list of statements or negated statements. Your algorithm should identify all possible faults that can explain the given observations.

4. Consider a simple household refrigerator. Identify all the replaceable components in a refrigerator. List the ways in which the components interact with each other. Consider only the direct interactions, and ignore interactions via other components. Describe the functioning of a refrigerator using these. What problems do you foresee? Now that you have a model, can you use this model for troubleshooting the refrigerator?

11. Hruday: A Heart Disease Advisor

11.1 Introduction

In this chapter, we will examine a Heart Disease Advisor, designed to provide advice to a non-specialist user. The system is not designed to play the role of a cardiologist who provides a prescription, or specifies a course of treatment. It simulates a knowledgeable friend who asks a person, possibly suffering from heart disease, a few relevant questions and gives him relevant and helpful comments.

The Advisor, named Hruday, deals with four major types of heart diseases:

- Ischaemic Heart Disease (IHD)
- Rheumatic Heart Disease (RHD)
- Congenital Heart Disease (CHD)
- Cor Pulmonale (CARP)

Its domain of application can be further delimited as follows:

- It deals only with symptoms reported by the patient.
- It does not handle ECG, X-Ray or pathology data.
- It takes cardiac risk factors into account.
- It does not prescribe a course of treatment.
- It assumes that the patient has no earlier history of heart disease, except in one or two special cases.

Hruday has been developed using the expert system shell Vidwan [NCST, 1993], developed at the NCST.

The case study will highlight the following issues:

- Use of certainty factors

- Structuring the reasoning into a number of major steps
- Use of Vidwan features for the development of Hruday

The main value of the case study will be to present a significant example of an ES application. You will find in the Hruday knowledge base (see Appendix B), examples of the application of many different features provided by the expert system shell Vidwan. Because Vidwan provides only for the creation of rule based systems, Hruday is in this form.

Because Hruday is available in the public domain, in machine-readable form, it is easy to experiment with. One can modify its rules, and add to them. One can examine its rules to see how they are structured to meet different objectives. One can use it as a model to create advisory ESs in other domains.

11.2 The Development of Hruday

Hruday began as a class project in a course on expert systems. One of the reasons for selecting this topic was to demonstrate that a KE can cope with specialist knowledge in a field initially strange to him, that he can identify and collect this knowledge, and structure it to be usable in an expert system. Of course, he demystifies the subject for himself in this process.

Expertise relating to heart disease is typical of expertise in general. The "common man" needs knowledge of heart disease to cope with life but, very often, lacks this knowledge. What an expert usually classifies effortlessly, the common man cannot. For instance, when is chest pain very significant and when is it not? The word *angina* means a lot to the expert, who uses the term to refer to pain having its origin in heart muscles in distress. He knows that it is characterised by increase of pain with physical effort, and is often accompanied by sweating.

Fifteen teams, with about six course participants in each team, worked on the project, relatively independently. They studied a number of books such as [Julian, 1983], [Wallwork and Stepney, 1987] and [Alpert and Rippe, 1980]. They interviewed physicians they knew.

Hruday was finally produced by the staff who had taught the course and had shared the project experience with the students they had supervised.

As an early project to use Vidwan, Hruday also contributed to Vidwan's development by giving useful feedback. For instance, a very early version of Vidwan (circa 1988) did not offer the *special, p* type parameter. Sex was, therefore, a boolean: male - true or false. So, when someone answered a question about his sex, typing in, say, "male", Hruday would promptly come back to ask "How sure are you?"!

Vidwan has a come a long way since then, and so has Hruday!

11.3 Some Domain-Specific Information

Congenital Heart Disease (CHD) is caused by defects one is born with, such as structural defects of the heart. CHD is usually detected early in life.

Rheumatic Heart Disease (RHD) is caused by damage to heart valves usually triggered off by repeated infections, often in childhood. This infection, named rheumatic fever, is usually treated with Penicillin or other similar antibiotics quite successfully. Repeated infections, if not treated properly, result in heart damage which usually manifests itself at a later stage in life.

Ischaemic Heart Disease (IHD) arises from reduced blood flow to heart muscles. It is usually found in people over 40 or 50 years old. A variety of risk factors, such as smoking, have been identified, which make a person more prone to IHD. A doctor would frequently use the fact that the person has more of these risk factors to conclude that suspected IHD is in fact present. Tests such as electro-cardiography under stress (produced by exercising on a treadmill) are valuable in confirming the presence of IHD in a very reliable manner.

Cor Pulmonale (CARP) is caused by a condition which involves the lungs as well as the heart. It is aggravated by intake of alcohol.

Because careless use of medical information may mislead a person, medical ESs usually carry appropriate warnings. Please see the warning given by Hruday, indicating that it is not meant for treatment, particularly self-treatment!

11.4 The Structure of Hruday

The complete listing of Hruday is given in Appendix B. The knowledge base begins with a set of type statements defining the type of the 54 attributes used. A few examples of different types of attributes are given below.

- Boolean (true or false attributes): managerial-job, smoker, epileptic etc.
- Multi-valued: suspected-disease, disease, complaint, etc.
- Special: sex (male/female), diabetic (yes, no, unknown), etc.
- Numeric: age, number of cigarettes per day, etc.

The system associates a certainty factor (CF) with the value of every Boolean attribute and every multi-valued attribute. So, if you say you are an epileptic, it asks you how sure you are of this fact. The CF you assign should be in the range from $+1$ (certainly true) to -1 (certainly false). The *special* type attributes are like *scalars* in some programming languages. They take one of the values specified in the corresponding *menu* statement.

There are 52 templates each associated with specified attributes. There are also 99 rules.

Examples of template and menu statements follow:

```
template(number_of_cigs, "How many cigarettes do you smoke per\n
        day, on the average?").
menu(complaint, [swelling_of_feet, tiredness, black_out, chest_pain]).
```

One of the simple things to notice here is the use of \n to control line breaks in the output of Vidwan. The notation interpreted by the shell permits you to do this.

In addition to the rule base file, Hruday uses another file which contains a template for presenting the conclusions of a consultation to the user. This uses the report generation facility of Vidwan explained in Appendix A.

11.5 Structure of the Reasoning Process

The best way to plan and create a hundred or more rules systematically is to have a structure for the whole process of diagnostic reasoning. No particular structure at this (higher) level is imposed on the knowledge engineer (KE). Vidwan allows him complete freedom to invent and use any such structure. The particular structure shown in Figure 11.1 is meant to serve as an example. The reader is invited and challenged to invent and try out other similar structures.

The backward chaining technique used makes the last step the top-level goal. The system's task is to find advice appropriate for the patient. The first few rules indicate what advice is to be given, each assuming that a specified disease is found to be present. Hence, the top-level sets up *disease* as a sub-goal, making it necessary for the system to conclude about the disease present. The rules which conclude about disease generally involve a suspected-disease and confirmatory conditions. Hence, the rules trying to determine disease set up *suspected-disease* and *confirmatory evidence* as sub-goals. In turn, the sub-goal *suspected-disease* sets up two sub-goals: *card-symp-present* and *IH-risk*. These sub-goals do not need further sub-goals, and are answered by direct questions put to the patient.

The reader is invited to create a small trial rule-base at this stage, to try out this recursive process of goal-finding. He should pay particular attention to the order in which questions are presented to the user.

The reader should also examine Hruday's rule-structure in the light of the need to ask the patient questions intelligently. As discussed in Chapter 6, appropriate filtering is carried out to ask questions only when they are relevant, and to ask them in the right order.

A good example of a rule with filtering conditions is cf11:

```
rule (cf11, HD_confirmation, true, 0.20) :-
    is(complaint, swelling_of_feet),
    is(sex, female),
    gt(age, 15.0),
    lt(age, 45.0),
    isnot(pregnant, yes).
```

Let us look at the idea embodied in this rule. Swelling of the feet may indicate heart failure, but swelling of feet in pregnancy may confuse us, not usually being of cardiac origin. So, we must ask the patient with swelling in the feet about the possibility of pregnancy. But, before asking the question, we must take into account the sex of the patient, the fact that she may or may not be in the reproductive age group, and then only should ask the question.

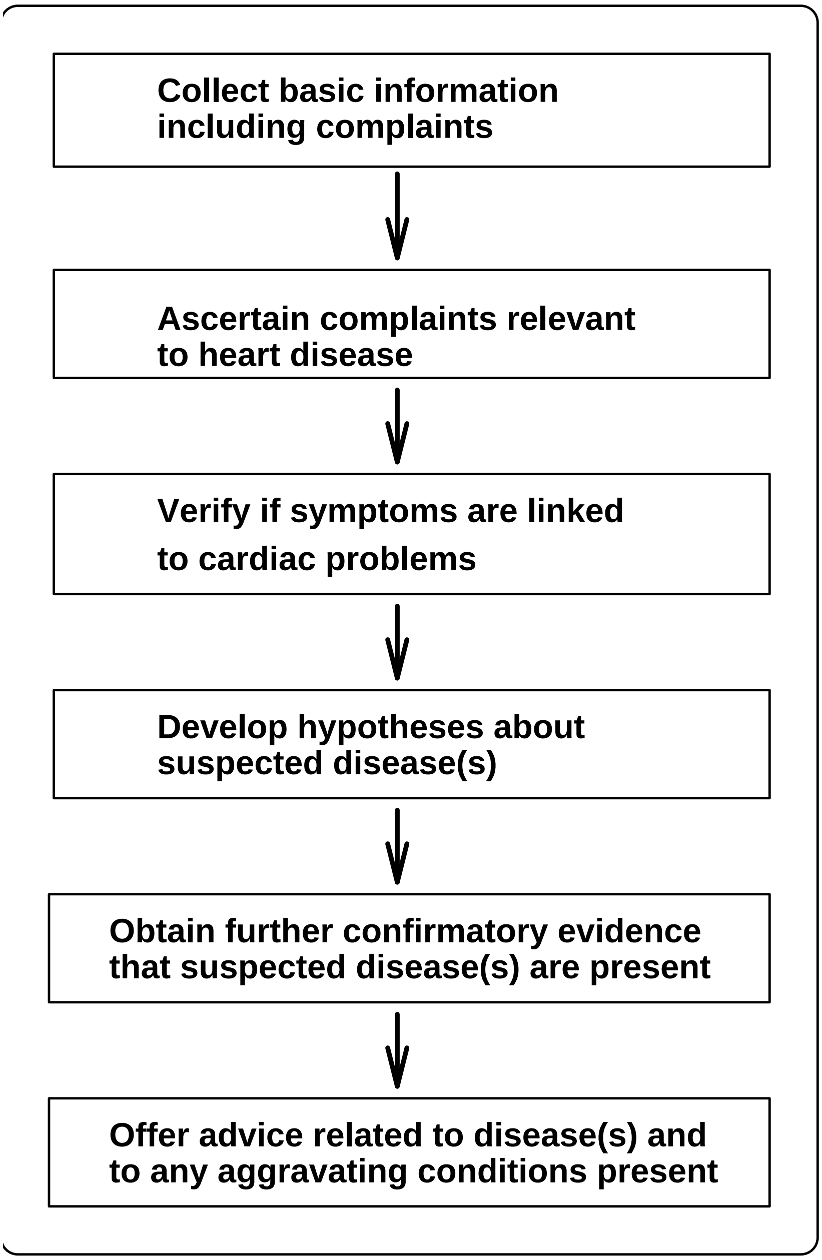

Figure 11.1: Structure of Reasoning Process in Hruday

No doctor can ask a 60 year old man if he is pregnant, without being laughed at! Expert systems have to be equally careful about the danger of asking such stupid questions!

11.6 Use of Certainty Factors

KEs should not normally expect to change the way a domain expert (DE) thinks about problems in his own field. For instance, you cannot normally expect a medical doctor to tell you what CF you can attach to the rule: *cigarette-smoking leads to the risk of heart disease.*

Indicative numbers are available in medical textbooks, but doctors do not use these numbers in diagnosing disease. Therefore, one should use statistical, probabilistic and numerical aids to reasoning very carefully. The way CFs are used in Hruday ensures that assignment of precise CF values is not important. The conclusions arrived at by Hruday do not change if you make small alterations to the CFs associated with the rules.

In the following sections, a few types of knowledge have been identified where the CF assignment can be systematised.

11.6.1 Sufficient Evidence

Consider the following example from outside cardiology:

> *Presence of the Koplik spot indicates measles*

Here, one can assign a CF in excess of 0.20 to the conclusion. If the user is sure of the antecedent (Koplik spot), the system has good reason to believe the conclusion. The general principle which is used is shown in Figure 11.2a.

11.6.2 Contributory Evidence

Consider the example:

> *Presence of fever contributes to finding of measles*

The antecedent *fever*, by itself, does not lead one to believe in the conclusion. But if there are other items of evidence in favour of the hypothesis, then this antecedent adds to the overall confidence in the conclusion. The CF for rules like this should be less than 0.20. Such rules in Hruday usually carry the CF 0.19.

11.6.3 Two Out of Three Items Necessary

A common situation in medical ESs requires that two or more items of evidence be treated as strong evidence to indicate the presence of a disease, while one item alone is treated as inadequate evidence. How do we represent this knowledge?

If each item of evidence contributes a CF of, say c, we need the following conditions to be satisfied:

- c < 0.2
- nadd(c, c) > 0.2

where nadd(c,c) is the value of the CF obtained by the non-linear addition of c each from two items of evidence. The general principle used when you need atleast two rules to fire in order to obtain sufficient evidence to make a conclusion is shown in Figure 11.2b.

The non-linear addition function you will find in Vidwan for two positive values of c is (see Section 5.5.2)

$$c + c * (1 - c)$$

If we choose c to be 0.19, then nadd(c,c) turns out to be 0.34.

nadd(nadd(c,c),c), which is the added value of CF from three items of evidence, is equal to 0.46.

11.6.4 Three Items Necessary

Consider the case where three symptoms are required to conclude something. In this case, we require that

- c < 0.20
- nadd(c,c) < 0.2
- nadd(nadd(c,c),c) > 0.2

c = 0.1 satisfies these conditions. c = 0.08 also does. In Vidwan rules, any value of c in the range $0.072 \leq c < 0.11$ will meet the needs mentioned above. That is, if there are a number of rules each of which can individually assign a value "c" to the CF, three of them together (not two) will contribute a CF $\geq$ 0.2.

11.6.5 The Problem with a Long Chain of Evidence

The CF contributed by the firing of a rule is $(a * r)$, where a is the CF associated with the antecedent, and r is the CF associated with the rule. Consider a long chain of rules like this:

If A **Then** B (CF = 0.3)
If B **Then** C (CF = 0.3)
If C **Then** D (CF = 0.3)

If the CF associated with A is 0.5, the CF associated with the other propositions will be as follows:

- CF associated with B = 0.15
- CF associated with C = 0.045

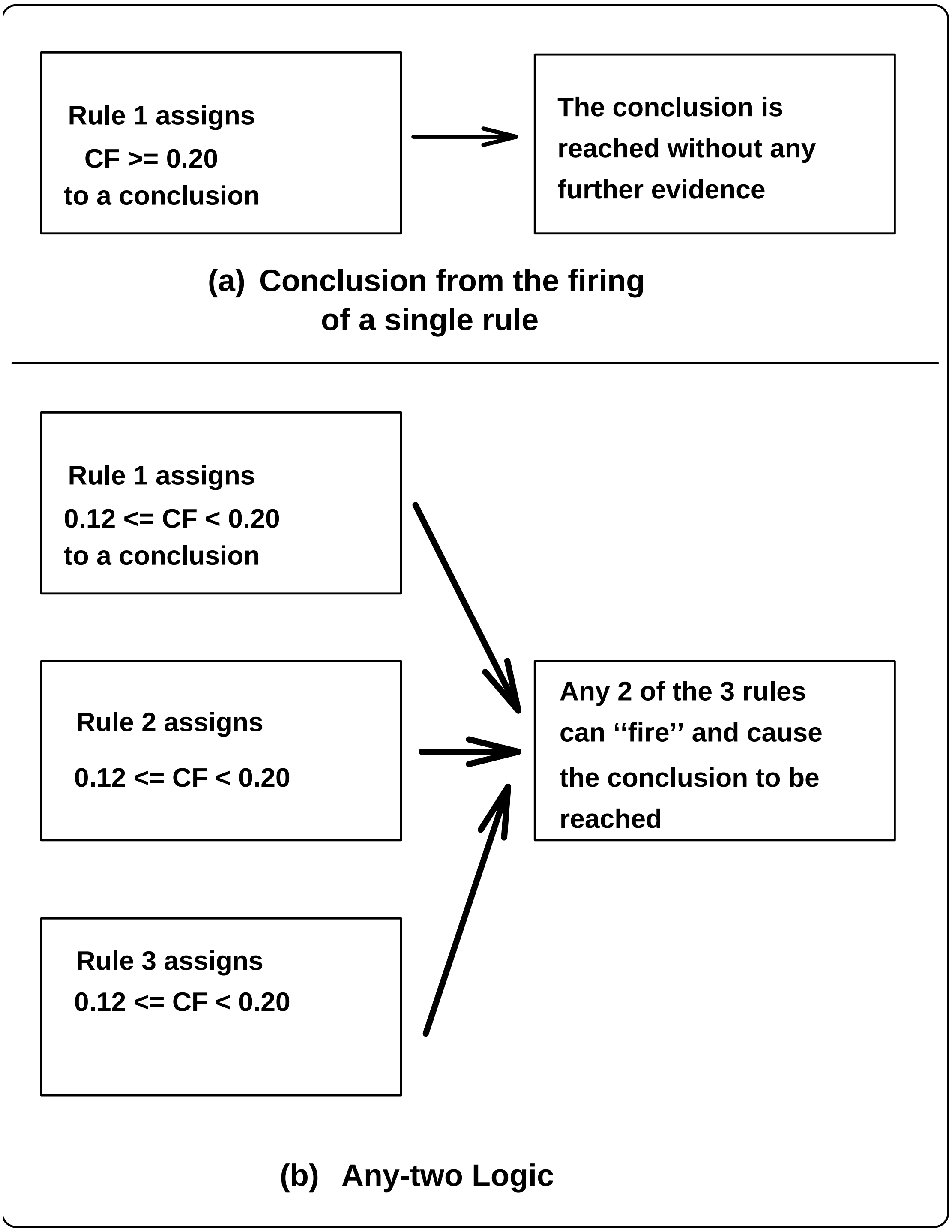

Figure 11.2: Critical Values of Certainty Factors

- CF associated with D = 0.0135

This indicates that one should be wary of having a long chain of reasoning, where the CFs associated with the rules are small. In this case, for instance, the second and third rules will not fire at all, because the CFs associated with B and C are too small. Also it can be verified that, even when the CF associated with A is 1.0, the third rule has no chance of firing.

One way of avoiding this problem is to avoid long chains of reasoning wherever possible. Another way is to pay special attention to the CFs and ensure that they are big enough lower down in the chain.

11.7 Other Issues Worth Noting

11.7.1 Repeated Consultations

A user may go through a question-answer session and see the results. Then, he may wish to modify the answer to one or more questions and see how the advice of the ES changes. This is valuable to the user. The facility that Vidwan provides for storing responses from one session, and of re-using them in another is relevant here. This facility also helps the KE to debug the ES, by repeated probing of its behaviour with a particular type of input.

11.7.2 Report Generation

Another feature of Vidwan worth commenting on is that of automatic *report generation*. You can modify the knowledge base of Hruday to create a report in simple English, at the end of a consultation session. Using values of attributes, given by the user as well as inferred, the system can offer the user its conclusions in a readable and convenient form.

A report generated by a sample run of Hruday is given in Appendix B.

Summary

- Hruday is a heart disease advisor implemented in Vidwan. It is meant to act as a knowledgeable friend who can offer suggestions and comments to users regarding heart diseases.

- Hruday makes good use of the CF method for representing various types of knowledge (such as m symptoms necessary).

- The reasoning process consists of gathering information (low level attributes such as complaint, age, etc.), checking for the possibility of heart disease (involving attributes such as card_symp_present and IH_risk), developing hypotheses (the relevant attribute is suspected_disease), confirming hypotheses (the relevant attribute is disease) and offering advice (the attribute is the goal attribute - adv).

12. GoodShow: A Television Fault Diagnosis System

12.1 Introduction

Expert systems technology has been applied in several tasks such as fault diagnosis, design, interpretation, etc. Several expert systems have been developed for the task of fault diagnosis of equipment, see, for example, [Rabinowitz et al, 1991] and [Register and Rewari, 1991]. The task of fault diagnosis can be described succinctly as follows: it is the task of identifying the faulty component(s) of a malfunctioning equipment, given the observed faulty behaviours (i.e., *symptoms*), the machine's responses to certain test patterns (i.e., *findings*) and other relevant information. Usually, a human expert carries out the task of fault diagnosis using measuring instruments and other tools. Most of the cases the expert encounters are mundane and repetitive in nature. Only a few cases are complex. Solving these few cases might require very specific knowledge and judgement acquired over a long period of time.

An expert system for troubleshooting should aim to solve all the mundane and less complex problems. The system should be able to identify problems of higher complexity to refer these cases to the human expert. Thus, the aim is not to build an expert system which can replace the expert who does the fault diagnosis. It is to free the expert from wasting his time on mundane problems and allow him to concentrate on the tougher problems. The expert is thus able to spend his time for more useful purposes.

Fault diagnosis is required for any type of equipment: mechanical, electrical or electronics or any hybrid system. One of the problems encountered when a machine breaks down is that the expert may not be available at the site at that time. In such cases, it is useful to have an expert system which could be used by an operator or lay person at the site. The system can also be used for the purpose of double checking the expert's diagnosis. An error by the expert due to carelessness or because certain facts have been overlooked can be detected. Any mismatch between the conclusions of the expert and the system may make the expert examine and possibly revise his diagnosis.

In this case study, we examine a television fault diagnosis system named GoodShow. The GoodShow system is a prototype expert system. We study the GoodShow system to understand the kind of real world fault diagnosis problems which can be solved using expert systems technology. We study how the fault diagnosis knowledge is encoded using rules, how the fault diagnosis strategy is implemented and how uncertainty is modelled in this domain.

12.2 The Domain

The domain of TV repair is a typical fault diagnosis task. The problem can be viewed at different levels of abstraction. It can be viewed either at the level of replaceable components or at the level of functional modules (high level).

At the component level, the TV set consists of IC chips and circuitry for receiving and processing video and audio signals, components of the speaker and the picture tube. The fault diagnosis task, in this case, involves the identification of chip(s) or component(s) which have to be replaced to repair the set. The process requires obtaining certain information by opening up the set and making measurements (perhaps using a multimeter) at certain points in the circuitry. However, the process requires a high level of expertise of the intricacies of the set. An expert system built out of knowledge at this level could only be used by an expert.

The TV repair task can also be viewed at a much higher level, that is, at the level of functional modules. The TV set can be viewed as having a number of interconnected functional modules. The troubleshooting task, in this case, is essentially to identify the faulty module(s). Of course, the whole module cannot be replaced since that will be unduly expensive. So, the complete job is not done. But automating even part of the job is useful. The GoodShow system is built using knowledge at the module level.

12.2.1 Modules, Faults and Observable Behaviours

In this section, the fault diagnosis knowledge about a TV set is described in terms of high-level modules (For example, sound processing circuit, picture tube, etc.) and symptoms which can be observed without using any tools (For example, picture jumps, sound is distorted, etc.).

A TV set consists of the following modules: antenna-and-tuner, sound processing circuit, intermediate frequency amplifier, colour processing circuit, synchronisation circuit, illumination processing circuit, picture tube, horizontal deflector of the picture tube, vertical deflector of the picture tube, speaker and internal power supply.

Broadly, the fault diagnosis knowledge can be described as follows: The antenna and the tuning circuit are used for tuning. If the antenna is not pointing in the right direction, then the screen could be full of tiny bright spots. The set may not produce any sound or it might produce a continuous hum. Improper tuning might result in either pale or intermittent colour pictures. If the antenna is not properly installed, then the picture will be jumping at irregular intervals. Physical obstructions to the reception by antenna could result in shadows of the images appearing.

For the set to work, it should get power from the plug point. The set has an internal power supply system. The panel lights of the set will be on only when the set gets supply from the internal power supply.

The sound quality will be bad if the sound processing circuit is faulty. A faulty sound processing circuit will cause the sound to distort either continuously or at intervals. If the sound is completely absent, it could mean that the sound processing circuit is not functioning. On the other hand, it could just mean that the speaker needs to be repaired; this is possible when the picture display is good.

Sometimes, the received signal might pick up interference due to a number of reasons, resulting in irregular distortion of both sound and picture. Any interference can also result in the display of a network of lines or dots.

There can be something wrong with the picture tube. The usual life span of a picture tube is eight years. After that, the brightness of the picture could go down. In general, a very low brightness suggests that there is something wrong with the picture tube. A problem with the contrast or brightness of the picture might indicate that the illumination processing circuit is faulty. A very high contrast almost certainly indicates that the illumination processing circuit is faulty.

If the colour does not show in the display, then it is possible that the colour processing circuit is faulty. Sometimes, a faulty colour processing circuit results in pale colours, intermittent colours or incorrect colours (For example, the hair of a person appearing blue). If the set is magnetised for some reason, then it usually results in the display of a patch of colour.

If there is something wrong with the horizontal deflector system, this can result in a number of faulty behaviours. It can result in a blank screen. Sometimes, the display is full of black and white lines or bars. A faulty horizontal deflector can result in a problem with the brightness of the images. It can cause the width of the images to be either inflated or shrunk. It can make the picture roll horizontally or shift the picture horizontally. A faulty vertical deflector can cause similar problems. It can cause the picture to roll vertically or to shift vertically. The height of the picture can be lengthened or shortened because of a faulty vertical deflector. It can also cause the picture to jump continuously.

A faulty intermediate frequency (IF) amplifier can cause a faulty picture display. This can result in a problem with the contrast of the picture. It can cause colour pictures to appear as black and white pictures or result in no image being displayed at all. If the screen shows just plain white light, then it surely suggests that the IF amplifier is faulty.

12.3 Rule Base Encoding

GoodShow is implemented using the rule based expert system shell called Vidwan (see Appendix A for a description of Vidwan). The complete listing of GoodShow rule base and log of a run of the system are given in Appendix C.

The facts in this domain are viewed as attribute-value tuples. If all the conditions in the condition part of a rule can be proved to be true, the system deduces that the conclusion of the rule is true. The fault diagnosis knowledge described in the previous section is encoded using rules. One such rule is illustrated below:

```
rule(id49, faulty_module, vert_deflect, 0.7) :-
    is(jumping_problem, true),
    is(type_of_jumping, continuously).
```

The rule can be roughly translated into pseudo-English form as:

rule id: id49
If there is a jumping problem
and the jumping is continuous
then the vertical deflector is faulty with CF = 0.7

The *CF = 0.7* in the conclusion part determines the measure of confidence in the rule. The CF stands for confidence factor. See Chapter 5 for more details on confidence factors.

In the rule, *id49* is the unique identifier for the rule. As is required for the Vidwan rules, the conclusion part has one conclusion (i.e., *faulty module is vertical deflector*). The conclusion has *faulty module* as the attribute and *vertical deflector* as the value. The attribute *faulty module* can take a number of values such as *vertical deflector, horizontal deflector, IF amplifier, colour processing circuit,* etc. – that is, all the modules under consideration. Also, a number of rules can come to the same conclusion, saying that the fault is with one particular module. In our rule base, the rules *id23, id44, id47* and *id49* all come to the conclusion that the *faulty module is the vertical deflector.*

The condition part of the rule has two conditions: *jumping problem is true* and *type of jumping is continuous*. The attribute *jumping problem* is of type boolean, which means that it can be either true or false. In the rule under consideration, the attribute *jumping problem* has to be proved to be true for the rule to fire. The next condition is that the *type of jumping is continuous.* In our domain, the attribute *type of jumping* can take one of the following values: *continuous* or *irregular*. In rule *id49*, the attribute *type of jumping* is checked to see if it has the value *continuous.* Thus, if one can prove that *jumping problem is true* and *type of jumping is continuous* then one can conclude that the *vertical deflector is faulty.* Note that the inferencing mechanism of Vidwan ensures that the condition *jumping problem is true* is verified before *type of jumping is continuous* is considered for verification.

Conceptually, the GoodShow rule base can be viewed, as consisting of a number of components.

First, we have a set of top level rules to determine the faulty module(s). All these rules have the attribute *faulty module* in the conclusion part.

There are a few rules to determine if the display is showing any illumination at all or if it is completely blank.

Similarly, there are a few rules to determine if the set is producing any sound at all.

In our domain, the fact that the display is showing lots of tiny dots is an important piece of knowledge. A few rules conclude this fact from the user responses.

A few rules determine if the TV set is dead – that is not producing any sound or picture.

While building rule bases for fault diagnosis tasks, it is useful to have rules for deducing *suspected* faulty parts. In our rule base, there are seven sets of rules focusing on each of the following (seven) potentially faulty modules: illumination processing circuit, picture tube, sound processing circuit, colour processing circuit, IF amplifier, synchronisation circuit and antenna-and-tuner.

The picture on the screen can have different types of problems which are determined by a set of rules.

There are rules for determining if the sizes of the images are improper and if the images are shifted.

Totally, the rule base consists of 73 rules. These rules are formed out of 50 attributes. Of these 50 attributes, the values of 23 attributes have to be obtained from the user and the other 27 have to be inferred from rules.

12.4 Fault Diagnosis Strategy

The task of the GoodShow system is to identify the faulty module(s) from the observable symptoms. For any such task, a certain number of important symptoms can be identified which act as the starting point for the fault identification process. If these *initial symptoms* are not identified then we can say that there are no identifiable faults. In our case, the initial symptoms are: *faulty picture* and *faulty sound.* Some module(s) are assumed to be faulty only when at at least one of these symptoms is observed. The GoodShow system is then used to locate the faulty module(s).

The presence of a particular initial symptom indicates the possibility of certain modules being faulty. The system then obtains other symptoms/information relevant to suspecting certain modules being faulty. The suspected faulty modules will be a subset of the set of possibly faulty modules indicated by the initial symptoms. The system then obtains more specific symptoms/information to confirm if they are indeed faulty, and to eliminate the possibility of other modules being faulty.

At the implementation level, the system tries to verify all the modules, one by one, (sequentially) to see if they are faulty. Certain modules are likely to be faulty only if the picture is faulty. Certain other modules are likely to be faulty only if the sound is distorted. For some other modules, both should be faulty. For example, if the only initial symptom exhibited is that the sound is faulty, then it means that the modules to be considered are the sound processing circuit and the speaker, or that there is some possible interference. At this point, the system acquires more symptoms/information to determine the suspected faulty modules. In our case, the system finds out if there is no sound or if the sound is distorted. If the user's response is that there is sound but is distorted, then the system suspects the sound processing circuit and the possibility of interference; it rejects the speaker from the set of possible faulty candidates. Now, the system obtains some more relevant symptoms/information to confirm if the fault is really with the sound processing circuit or if there is a possibility of interference. That is, the system finds out if the sound distorts along with the picture. If so, it means that there is an interference problem. On the other hand, if the sound distorts on its own and the picture is stable, then it can mean that the sound processing circuit is faulty. Assuming that the user says that the sound distorts on its own and the picture is stable, then the GoodShow system concludes that the fault is with the sound processing circuit.

12.5 Modelling Uncertainty

In general, the knowledge used for the task of equipment fault diagnosis is more or less certain. There is not much room for modelling uncertainty. In our rule base, most of the rules have a high confidence factor associated with them. For rules which are certain or fairly certain we have given CFs ranging from 0.7 to 1.0. There are rules which are suggestive of the conclusion; for these rules we have given CFs ranging from 0.4 to 0.6. Some rules are indicative of the conclusions; for them we have given CF of 0.25.

Consider the rule we referred to in the Section 12.3 on Rule Base Encoding. The pseudo English translation of the rule is given below:

rule id: id49
If jumping problem is true
and type of jumping is continuous
then faulty module is vertical deflector with CF = 0.7

The conclusion part has CF = 0.7. In Vidwan, the uncertainty in the domain knowledge is modelled using a number in the range from −1.0 to 1.0. A CF of 0.7 says that the rule can be believed with a fair degree of certainty. In other words, it says that the conclusion *faulty module is vertical deflector* can be considered true with a fair degree of certainty when the conditions *jumping problem is true* and *type of jumping is continuous* are certainly true (i.e., have CF = 1.0). Of course, if the conditions are not certainly true, then the measure of confidence in the conclusion should also go down from fairly certain to some thing less. The uncertainty handling mechanism in Vidwan provides a procedure to determine the measure of confidence in the conclusion based on the confidence of the conditions.

The knowledge encoded in the rule id49 is almost sure. Rarely we do encounter a case where the picture is jumping continuously but the vertical deflector is in good working condition. These cases could be accounted for by some problem with the signal or reception circuit or processing circuit. We have represented the uncertainty in the knowledge using the number 0.7. A question might arise why it should be 0.7? In fact, it need not be exactly 0.7. The main point is that it has to be close to 1.0, because the knowledge is fairly certain and at the same time, it should not be 1.0 because the knowledge is not definitely certain. It could be 0.8; on the other hand if it is 0.5, then it misrepresents the measure of certainty associated with the knowledge.

Some of the symptoms could indicate a problem with a number of modules. A problem with any one of these modules could result in the manifestation of the symptom, but this is not certain. In the GoodShow system, one such symptom is *problem with the brightness*. A problem with the brightness of the picture can possibly indicate a problem with the picture tube, horizontal deflector and illumination processing circuit. These multiple possibilities and the corresponding uncertainties are reflected in the following rules: id40_2, id39 and lpc2. The pseudo English translation of these rules are given below for ease of reference.

rule id: id40_2
If brightness problem is true
then suspected faulty module is picture tube with CF = 0.7

rule id: id39
If brightness problem is true
then faulty module is horizontal deflector with CF = 0.25

rule id: lpc2
If brightness problem is true
then suspected faulty module is illumination
processing circuit with CF = 0.5

These rules encode the knowledge that: if *brightness problem is true* then it suggests that the suspected faulty module is illumination processing circuit or picture tube or this is an indication that the faulty module is horizontal deflector.

According to rule id40_2, if brightness problem is noticed, then there is a suspicion that the picture tube is faulty. Consider the rule id40_1 whose pseudo English translation is given below:

rule id: id40_1
If suspected faulty module is picture tube
and screen displays pictures and not tiny spots
and brightness is very low
and TV is at least 8 years old
then faulty module is picture tube with CF = 0.7

This rule is applicable if we already suspect that the fault is with the picture tube. It states that if we know that the age of the TV (or picture tube) is greater than 8 years and the brightness is too low, then we can say fairly certainly that the fault is with the picture tube. This rule shows how a suspicion that the faulty module is picture tube is being confirmed due to the availability of some more evidence.

Summary

- The GoodShow system illustrates how a rule based expert system can be built to perform the task of fault diagnosis.
- The system can be used by any lay person who owns a TV to determine what is wrong with the TV whenever there is some problem with either the picture or the sound.
- The rule base consists of knowledge to identify the faulty modules, but does not include knowledge about how to identify the faulty components which could be replaced to rectify the fault.
- The knowledge is encoded as a set of rules. The rule base can be conceptually divided into a number of components.
- The process of fault diagnosis involves finding out if the initial symptoms are present; determining the suspected faulty modules and confirming the faulty modules.

- The uncertainty in the knowledge can be classified into three levels: rules confirming some conclusions, rules suggesting some conclusions and rules indicative of some conclusions.

Acknowledgement

The initial version of GoodShow was developed by Atul Hatalkar, Durgesh Rao and Mangesh Risbud, participants in the Post Graduate Diploma in Software Technology offered by the National Centre for Software Technology in 1990, as part of their course work.

13. ConTax: An Income Tax Planning Consultant

13.1 Introduction

Income tax planning deals with determining ways of utilising the income of a person to minimise the amount of tax payable. The tax laws specify the amount of tax chargeable on the income of a person. They also specify certain tax reduction mechanisms. These are in the form of rebates from tax, deductions from chargeable income and exemptions from tax for certain types of income. The purpose of income tax planning is to reduce the tax chargeable as much as possible. This planning is done taking into consideration the available tax reduction mechanisms and the tax payer's requirements and preferences.

This planning can be done by expert systems. In fact, expert systems technology is being applied increasingly in the domain of income tax planning [Sasikumar and Ramchandar, 1992]. There are a number of reasons why expert systems technology is suitable for application in this domain.

Firstly, a large number of people are concerned with this problem, but the expertise is not widely available. The expertise is available with the professionals. They need to be hired for a fee. The expertise is also available to a certain extent in books. But consulting these books takes time and effort. Usually people find this demanding. Thus, most people do not bother too much about tax planning. Most of the planning they do is through informal consultations with knowledgeable friends. As a result, they usually end up paying more than the minimum amount of taxes due from them. Having expert systems in office environments would provide professional guidance to tax planning for a large number of people. The cost of such a guidance would be less compared to the fee payable to professionals. The effort required would also be less compared to acquiring the expertise themselves from books.

Another reason for using expert systems for income tax planning is its ease of modifiability. Tax laws change frequently. The knowledge base of an expert system can be appropriately modified to incorporate any new piece of knowledge or any updates. Thus the ease of modifiability is very useful for the domain of income tax planning.

Also in a modern office environment, tax related information of all employees is available on-line on computers. This facilitates building an expert system for income tax planning as most of the required data is available on-line.

At the National Centre for Software Technology (NCST), Bombay, an expert system called ConTax [Ramchandar, 1992] has been developed. This system has been developed with the aim of assisting employees of NCST in their income tax planning. The system is designed as an integrated system with three expert system modules and one spreadsheet-like module embedded in it. At present, the ConTax system is at the prototype stage. At NCST, the accounts division maintains the salary information of all employees in an on-line database. The ConTax system can, in principle, use this database to obtain salary information. Future work is envisioned to put the system to use on a regular basis within NCST.

In the next section, we describe the domain considered for ConTax. In Section 13.3, we analyse the ConTax system as an integrated system consisting of embedded systems. Next we study the four embedded modules in some detail. This is followed by a section on the future work envisioned and a summary.

13.2 The Domain

The ConTax system deals with the domain of income tax planning. Its target user group is employees of NCST. The salary of the employee is the primary source of income. It consists of basic pay, dearness allowance, house rent allowance, city compensatory allowance, shift allowance, etc. The amount of tax chargeable on an individual depends on the slab into which the individual's annual income falls. The rates of income tax in the case of individuals are as follows (for the assessment year 1992-93):

Slab of Total Income	Rates of Income Tax
Upto Rs 28,000	Nil
Rs 28,001 to Rs 50,000	20 percent of the amount by which the income exceeds Rs 28,000
Rs 50,001 to Rs 1,00,000	Rs 4,400 plus 30 percent of the amount by which the income exceeds Rs 50,000
Above Rs 1,00,000	Rs 19,400 plus 40 percent of the amount by which the income exceeds Rs 1,00,000

A standard deduction is applied to the annual income before determining the applicable slab. The standard deduction is calculated as the minimum of the standard deduction ceiling and 33 percent of the annual income. The standard deduction ceiling for a female who earns upto Rs 75,000 is Rs 15,000; for others it is Rs 12,000.

Apart from the standard deduction, there are a number of other deductions which can be applied to the annual income before determining the applicable slab. These deductions are applicable when the individual opts to invest in certain schemes. The investment schemes considered include: Public Provident Fund, Unit Linked Insurance Plan, National Savings Certificates Issue

VIII, Equity Linked Savings Scheme, Compulsory Time Deposits, Jeevan Dhara and Jeevan Akshay. There are a number of factors which influences an investor's choice of investment schemes. These factors include security, lock-in period, possibility of selling the investment, possibility of premature encashment and possibility of utilising the investment for raising a loan.

The taxable income is calculated as the annual income minus all applicable deductions. Then the tax chargeable on taxable income is determined according to the slab in which it falls.

A rebate is applied to the amount of tax chargeable on taxable income. The amount of rebate depends on the amount of total deduction applicable.

Total Deduction	Rebate
Upto Rs 60,000	20 percent of the total deduction
Above Rs 60,000	Rs 12,000

Applying the rebate to the tax chargeable gives rise to a reduced tax. If the taxable income is greater than Rs 1,00,000, then a surcharge is added to the reduced tax to get the final tax. The surcharge is calculated as 12 percent of the reduced tax.

The information given above is for the assessment year 1992-93. The tax laws change usually every year, sometimes more frequently.

13.2.1 Sources of Expertise

Most of the expertise required for a system of this type is available in books. Tax procedures, investment schemes and tax deduction procedures are well documented. The books consulted during the development of ConTax include [Singhania, 1992], [Shanbag, 1991] and [Palekar, 1991]. But human expertise is not completely dispensable. To suggest an investment strategy as per the preferences of the user require a certain amount of subjective expertise. Professional tax consultants use this expertise to plan for their clients. To capture this kind of expertise an investment consultant was also interviewed.

In the next section, we will analyse ConTax system in terms of its inputs, its outputs, the embedded systems, etc. We also give a brief description of the implementation details of the embedded systems and communication among them.

13.3 ConTax as an Integrated System

The ConTax system can be called an integrated system as it consists of four embedded modules. The input and output of the integrated system are listed below:

13.3.1 Input

The input to the system includes the following:

- salary information
- personal investment preferences
- investments already committed
- loan repayment requirements

13.3.2 Output

The system generates a report as its output. It contains the following information: the salary information, tax deductions due to standard deduction and investments and the final tax payable by the employee.

13.3.3 Embedded Modules

The integrated system consists of three expert system modules and one spreadsheet-like module. The three expert system modules are:

- Taxable salary module
- Tax liability module
- Investment rating module

The spreadsheet-like module is:

- Investment allocation module

The taxable salary module computes the taxable salary from salary information. The taxable salary is passed on to the tax liability module; this module computes the final tax payable taking into consideration the taxable salary, investments committed by the user, rebate on tax, etc.

The investment rating module computes the ratings for the investment schemes under consideration. These ratings are computed using factors such as the user's preferences on level of risk involved, lock-in period, pay-back method, etc.

The investment allocation module takes its input from the tax liability module and the investment rating module. This module enables the user to experiment with alternative investment strategies and visualise their implications before committing to a particular strategy.

In the next section, we will see how command files are used to implement and integrate these modules.

13.3.4 Integration of Expert System Modules

The expert system modules are integrated using Vidwan command files (see Appendix A for details on Vidwan). Figure 13.1 shows the overall structure of ConTax system depicting its various modules and the direction of communication among them.

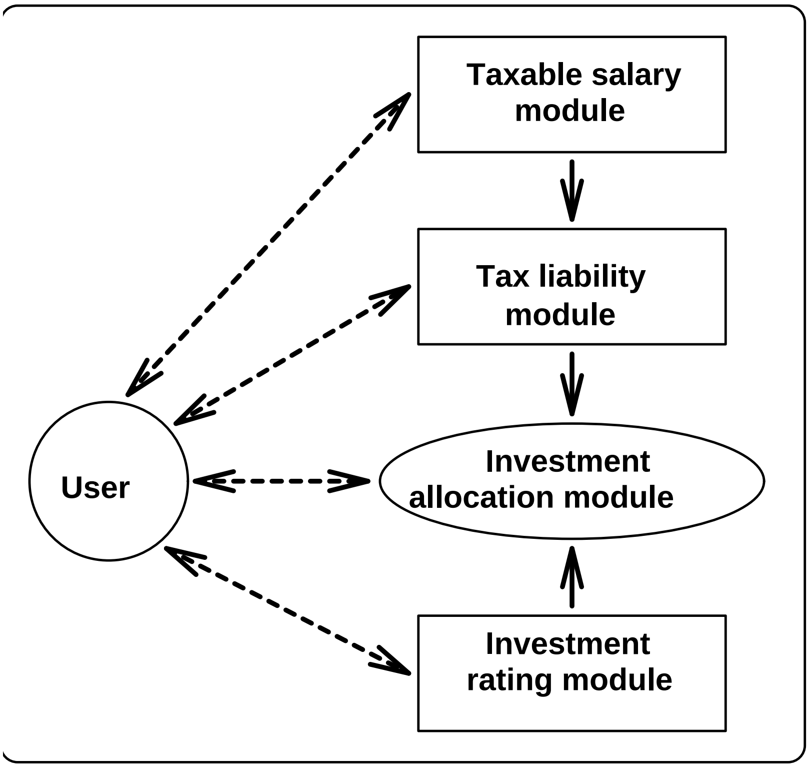

Figure 13.1: Block Structure of ConTax

Taxable Salary Module

The command file used by the taxable salary module is "tax_sal". This file consists of the following sequence of commands:

```
load tax_sal.rb
response tax_sal.res
run
report tax_calc.inp
printresults tax_sal.prn
saveresponse tax_sal.res
exit
```

The interpretation for these commands are given in the Appendix A on Vidwan.

A important thing to be noted is the file "tax_calc.inp". This file is created using the report generator facility. The data in this file include taxable salary, basic and dearness allowance. This data is passed on to the next module. First, this data is transferred to the file "tax_calc.res". If this data is already present in "tax_calc.res", then it will be updated. The command file for the tax liability module loads the response file "tax_calc.res". This way the data is transferred from the taxable salary module to the tax liability module.

The file "tax_sal.prn" will form the first part of the final report. This file contains information such as the total income including basic pay, house rent allowance, etc., and the standard deduction.

Tax Liability Module

The command file used by the tax liability module is “tax_calc”. This file contains the following sequence of commands:

```
load tax_calc.rb
response tax_calc.res
run
report tax_calc.out
printresults tax_calc.prn
saveresponse tax_calc.res
exit
```

This module generates the file “tax_calc.out” using the report generator facility. This file is taken as a input by the investment allocation module. The file contains information such as the taxable salary, tax deductions due to various investment schemes and the final tax. Details on how this information is being used by the investment allocation module are explained in Section 13.7 on Investment Allocation Module.

The file “tax_calc.prn” forms the second part of the final report. The file “tax_calc.prn” contains tax deductions due to investment schemes and insurance schemes, and the final tax. Thus the final report is the concatenation of the files “tax_sal.prn” and “tax_calc.prn”.

Investment Rating Module

The investment rating module makes use of the command file “planner”. The “planner” file consists of the following commands:

```
load planner.rb
response cust.res
run
report planner.out
saveresponse cust.res
exit
```

The investment rating module is used to determine the rating of various investment schemes. This module takes one investment scheme at a time and determines its rating. It outputs the name of the investment scheme and its rating into the file “planner.out”. This process is repeated for all the investments. Each time the file “planner.out” is updated, the data from it is collected into an array and finally the array is passed as input to the investment allocation module. Details of how the investment allocation module makes use of this array are described in Section 13.7 on Investment Allocation Module.

Investment Allocation Module

The investment allocation module provides a spreadsheet-like interface to the user. It lists all the investment schemes along with their ratings. It displays information on the money invested in each of these schemes, the total income, the tax deduction, the final tax to be paid, etc. The user can increment or decrement the investment amount in a scheme and see its implications on other parameters such as net tax. As noted earlier, the input data to this module is supplied by the files "tax_calc.out" and "planner.out".

The next four sections look at the embedded modules in some detail. The expert systems modules are explained with example rules.

13.4 Taxable Salary Module

This module takes the salary information as input and computes the taxable income of the person. First, the annual income is computed. This amount minus the applicable standard deduction amount is the taxable salary of the person.

This module is implemented using a rule base called *tax_sal.rb*. All the attributes used in this rule base are of numeric type. The arithmetic calculations required to calculate the annual income, standard deduction and taxable income are implemented using *compute rules*. The rule base has 12 compute rules. For instance, consider the following compute rule:

```
compute(std3, std_ded_ceiling, 12000) :-
        is(sex, female),
        gt(total_income, 75000).
```

This compute rule implements the following rule:

The standard deduction ceiling for a female employee whose total (annual) income is greater than Rs 75,000 is Rs 12,000.

13.5 Tax Liability Module

This module takes as input the proposed investments and other tax reducing loan repayments made by the person. It determines the reduction in the taxable income due to investments and other loan repayments. Then it computes the actual tax to be paid out of the reduced taxable income.

The tax liability module is implemented using a rule base called *tax_calc.rb*. Almost all the attributes used in the rule base are numeric. There are 33 compute rules and 35 attributes in this rule base.

The following example is one of the compute rules used in the rule base:

```
compute(tax2, tax1, (0.2 * (taxable_sal - 28000))) :-
    gt(taxable_sal, 28000),
    le(taxable_sal, 50000).
```

This compute rule implements the following tax rule:

If the taxable salary of a person is between Rs 28,000 and Rs 50,000 then the *tax to be paid by the person is 20 percent of the amount over Rs 28,000.*

13.6 Investment Rating Module

This module assigns a preference rating to the investment schemes from certain preference measures obtained from the user. These preference measures include risk preference, return expectancy, etc.

The rating for an investment scheme is determined by consulting a rule base. The file name of this rule base is *planner.rb*. It contains 22 rules and 21 attributes. To obtain a general idea about the determination of rating for a scheme from the user preferences, consider the following rule:

```
rule(ISR_JEEVAN_AK1, rating, true, 0.2) :-
    is(investment_name, Jeevan_Akshay),
    is(approaching_retirement, yes).
```

This rule encodes the following judgement:

Give a positive rating (0.2) to the investment Jeevan Akshay if the user is approaching retirement.

13.7 Investment Allocation Module

This module is implemented as a spreadsheet-like mechanism. All the investment schemes under consideration are listed in the descending order of their ratings. The user is shown the money invested currently in all these schemes, the total income, tax deduction, total tax payable, etc. The user is allowed to experiment with the amount invested in each of these schemes. Once the user enters how much is to be increased or decreased for an investment, the system computes its implications on all relevant parameters such as tax payable, tax deduction, etc. The user can repeat this experimentation several times before deciding on a particular strategy.

13.8 Future Work

The ConTax system is at present at the prototype stage. The system needs to be improved to the level where it will be used by the NCST staff to plan their investment strategies.

One of the things which need to be done is to enable the system to directly make use of NCST's salary database. This database is maintained using a standard relational database management system. This means that an interface has to be built to extract salary information of a particular user. This brings in the issues of data security and privacy.

At present, the annual income is calculated purely based on current monthly income. This is not always valid as, for example, it does not take into consideration a possible increment during the year. Thus the taxable salary module needs to be improved.

Summary

- ConTax is a prototype expert system on income tax planning.
- It is implemented as an integrated system with three expert system modules and one spreadsheet-like module embedded in it.
- The major sources of expertise are books on tax and tax deduction regulations.
- The system is a first step towards realising the goal of building an operational expert system for income tax planning to be used by the staff of NCST.

Acknowledgement

The ConTax system was developed by a student K Ramchandar, with guidance from M Sasikumar, as a partial fulfillment of the requirements for the degree of Master of Computer Applications offered by the University of Poona in 1992.

A. Vidwan: An Expert System Shell

Vidwan is an expert system shell developed at the National Centre for Software Technology, Bombay. It enables knowledge of a domain to be encoded in the form of if-then rules. It supports backward chaining as the inference procedure. It enables the user to represent the uncertainty associated with the domain knowledge using certainty factors. It provides explanation facilities such as *why* and *how*. Vidwan allows one to create a rule base using any standard text editor or the system's built-in interactive editor.

We briefly look at the main features of Vidwan and examine the syntax of its rule language. Some special features such as the report generator and command file directed invocation of Vidwan are also discussed briefly. For more details, please refer to the user manual of the software [NCST, 1993].

A.1 An Overview of Features

Prototypes have been developed on Vidwan for a large variety of applications including medical advisors (For example, cardiology, respiratory diseases, rheumatism, etc.), troubleshooting systems (electric motor, electronic equipments, printer, etc.) and financial advisors (credit worthiness, share investment, tax planning, etc.).

The basic components of a rule base for Vidwan are the parameters of interest in the domain (such as the age of the person, symptoms reported, etc.) and the way these parameters are related. The parameters of the domain are called *attributes* in Vidwan. The relations among them are represented using if-then rules. The next section explains the nature and the syntax of the rules as used in Vidwan. Apart from the specification of attributes and rules, Vidwan also allows the user to specify many other types of information in the rule base. These are used to mainly control the system's interaction with the user. These are also briefly covered. We also discuss some special features of Vidwan such as the ability to produce reports and to embed Vidwan calls in other applications programs. In the rest of this section, we briefly describe some main features offered by the shell.

Vidwan provides facilities to record user interactions in a file to obtain a *log* of a session. The use of *menus* and *templates* for attributes are supported. This can be used to make questions to the user sufficiently detailed and comprehensive. A limited arithmetic capability is also provided. Vidwan provides a friendly *user interface* using windows, cursor-selection of commands and on-line help.

Saving and loading responses: The user can save the responses given during a session in a file and load them during another run. This enables using standard stored test responses for debugging a rule base. This facility is also useful for continuing an earlier consulting session with revised data. Vidwan also provides a flexible *rule base browser* which can be used to examine the rule base and the status of various rules and conditions during a run.

Another useful feature is the "revoke" option. This option allows the user to *review the responses* he had given so far in the current run. This facility can be requested any time a question is posed to the user by Vidwan. When the system walks through the list of attributes and values one by one, the user can selectively clear the values that an attribute has been set to.

There is a general purpose *external program interface*, which can be used for database accesses, graphics support, automatic data acquisition, etc. Vidwan also provides a direct access to databases maintained as text files.

A built-in *report generator* facility is provided to design application dependent layouts for display of conclusions as well as for generating detailed reports of consultation sessions.

Vidwan applications can be embedded in programs developed in other languages. This is done by creating a command file for Vidwan and invoking Vidwan with the command file. In this case, Vidwan's actions are controlled by the command file, instead of being chosen by the user dynamically from menus.

A.2 Rule

The form (structure) of a Vidwan rule is shown below:

rule (<id> , <attr> , <value> , <cf>) :- <anteset>.

The symbols with the angular brackets indicate generic terms which will be described in the following paragraphs. All other non-blank characters are a mandatory part of the syntax. Each rule has a unique identifier, <id>. <attr> is the name of the attribute and <value> is its value. <cf> is the certainty factor, which can be associated with the attribute.

The <anteset> consists of one or more antecedents separated by commas. The last antecedent should be followed by a full stop. Each antecedent is of the form:

<opr> (<attr> , <value>)

where, <opr> is one of a pre-defined set of operators. For example, a rule from the Hruday (see Appendix B) rule base looks like this:

```
rule(chp2, chest_pain_type, cardiac, 0.25) :-
        is(complaint, chest_pain),
        is(chest_pain_location, left_side_of_chest).
```

This rule says that:

If the patient's complaint is chest pain
and the site of the chest pain is the left side of the chest
then there is suggestive evidence (0.25) that

the type of the chest pain is cardiac.

Each rule can have only one consequent. The antecedents of a rule are conjunctively connected (**and**ed) together. Rules can have zero or more antecedents. A rule with zero antecedents is applicable (i.e., can be fired) in all cases. (This can be used to define default values for parameters.) For example the following,

rule (bpl, bp-status, normal, 0.3).

says that unless modified by other rules, blood pressure status is normal with certainty 0.3.

In the description above, all symbols, except the symbols with angular brackets (generic terms), are part of the syntax and have to be typed in as given. One or more blanks are allowed wherever there is at least one blank in the description.

The core of an antecedent or consequent is an attribute-value tuple. In the example above, the attributes are *chest_pain_type, complaint* and *chest_pain_location* and the their values are *cardiac, chest_pain* and *left_side_of_chest* respectively.

A.3 Characteristics of an Attribute

Normally, for each attribute used in the rule base, a number of characteristics have to be defined. Some of these are required to provide a better user interface, so that the questions asked will be understandable to the end user of the system. The other characteristics contribute to the efficiency of the system. Knowing more about the characteristics of each attribute can help the system phrase its questions and prompts better. For instance, for an attribute with a numeric value, say, the age of the patient, there will usually be only one value; in contrast, for an attribute such as disease of the patient, there can be more than one value. More important, users are certain about their answers to questions like "What is your sex?", "What is your age?", etc. For these questions, the system should not prompt for the certainty factors to be associated with the values. On the other hand, for questions such as "Do you have breathlessness even while you are not taxing yourself physically?", the system should ideally ask for the certainty factor associated with the response (i.e., value).

type, template and *menu* are properties associated with an attribute. These are described in the following sections. Most of these properties are optional. If they are not specified, the system has defaults for them.

A.3.1 Type

Attributes can be classified into five types — boolean, numeric, special, multivalued and single-valued.

Boolean

A boolean typed attribute can have the value "true" or "false". In Vidwan, in order to evaluate a boolean attribute, the value of the certainty factor for the attribute is evaluated, if possible, or prompted for from the user. A positive certainty factor for the attribute indicates the value *true* and a negative certainty factor indicates the value *false*. If the certainty factor for the attribute is less than $+0.2$ but greater than -0.2, then the value of the attribute is considered to be neither true nor false.

Numeric

A numeric typed attribute takes a number as its value. It is assumed that values for numeric attributes hold with full certainty (i.e., certainty factor = 1) and hence no certainty factor is explicitly prompted for from the user. A numeric attribute can have only one value at a time. Arithmetic comparison is allowed only with other numeric typed attributes. The compute clause (see Section A.6) allows an arithmetic expression to be evaluated and the value to be assigned to an attribute. The attributes occurring in the consequent of a compute clause should also be numeric.

Numeric attributes are available in two formats: integers and floating point numbers. Integer-typed attributes allow only whole numbers as the value and any fractional part in the input is truncated. Floating point numbers allow any real number as the value of the attribute. For example, the number of completed years of a person in service will be an integer valued attribute, whereas the area of a figure will be a floating point number.

Special

The values of some attributes such as name and sex of an individual are always fully certain. Such attributes can be defined as "special" attributes. While evaluating a special typed attribute, only one value out of a set of allowed values is assumed to hold and a certainty factor of 1 is assigned to it automatically.

Multi-valued

A multi-valued attribute can take more than one value at a time, possibly, with varying certainty factors for each of the values. Attributes such as disease of a patient, countries in a continent, etc., belong to the class of multi-valued attributes. An attribute is multi-valued if it can have more than one value with full certainty.

Single-valued

A single-valued attribute takes only one value – if the value is known with full certainty (certainty factor > 0.95). But in case one value is not known with full certainty, a single-valued attribute can have more than one value with varying certainty factors. The type of a material, the colour of an object, etc. are examples of single-valued attributes.

Type Definition

A type definition of an attribute has the form

type(<attr>, <type>).

Where, <attr> is the attribute name and <type> can be any one of the following:

b	for boolean
i	for numeric (integer)

f	for numeric (floating point)
p	for special
s	for single-valued
m	for multi-valued

All type definitions should occur at the beginning of the rule base. And all attributes used in the rule base should be given a type definition.

A.3.2 Template

While trying to evaluate a particular attribute, if Vidwan notices that no rules or computes have been defined to deduce a value for that attribute, the user is prompted for its value. By default, Vidwan frames the prompt in a standard way. For boolean attributes, the following format is used:

How confident are you that <attr>?

For attributes of any other type (i.e., other than boolean), the format of the prompt is:

Enter the value of <attr>:

By defining a template for an attribute, a more meaningful prompt can be created, instead of the standard prompt.

The template should be a string within double quotes. A template can be defined for any attribute. For an attribute whose value has to be got from the user, a meaningful question can be defined as the template. For an attribute whose value can be deduced from the rules, a translation of the attribute name can be defined as the template. This latter type of template is useful to find the name of some attribute at run time for use inside the explanation mode. Vidwan provides a simple form of pattern matching on template definitions to help the user find out the name of an attribute that he is looking for.

An example of a template definition for an attribute is:

```
template(palpitations, "Do you sense your heart beating
             prominently even when you are not taxing
             yourself physically?").
```

Vidwan provides limited amount of run time formatting of templates. You can embed an attribute name inside the template string. When the template is used by Vidwan, the current value of the attribute is substituted in place of the name. For this, the attribute name should be enclosed inside a pair of "^" signs.

For example, a template such as "Your body temperature is ^temp^ which indicates high fever. Are you shivering?", could get displayed to some user as "Your body temperature is 102, which indicates high fever. Are you shivering?", if the attribute *temp* has obtained the value 102, before the question is asked.

Where there are more than one value for the attribute, the value with the highest certainty factor obtained will be used.

A.3.3 Menu

Validating the responses to a question poses a serious problem. Since Vidwan does not have any domain knowledge, normally no input validation can be performed. The type definition of an attribute allows some checking to be performed on the value of the attribute. For instance, a numeric attribute should have a number as its value.

For multi-valued, single-valued and special typed attributes, a limited form of input validation can be performed provided that Vidwan knows what values are possible for the attributes. A menu can be defined for each attribute, listing out the possible values. If a menu has been defined for an attribute, Vidwan will accept only values from the menu as a value of the attribute from the user. For boolean attributes, the value is always assumed to be true (so there is no menu) and only the certainty factor is prompted for. If menus are defined for an attribute, and if values other than those in the menu are being used along with the attribute in the rules, Vidwan will give warning messages.

Similarly, for many numeric attributes, acceptable values fall in a range which can be pre-defined. Such a range of values can be defined as the menu for a numeric attribute. Vidwan will then accept only values within the range (inclusive of the end points) as the value of the attribute, from the user.

A menu definition for an attribute, which is multi-valued, single-valued or special typed, contains the attribute name and a list of values. The value list is enclosed in square brackets with commas separating the individual values. For example,

```
menu(chest_pain_location, [left_side_of_chest,
        centre_of_chest, right_side_of_chest]).
menu(sex,[male,female]).
```

Menus for numeric typed attributes are specified by giving the endpoints of the range. For instance,

```
menu (systolic_blood_pressure, [100, 210]).
menu (haemoglobin_level, [1.0, 30.0]).
```

A.3.4 Note

At the end of a run, Vidwan generally displays as its conclusion, only the values that have been associated with the goal attribute. If more information or an explanation is required, Vidwan provides details through *why*, *how*, and other options. But often, some messages need to be displayed along with the conclusions. For instance, a system doing a drug selection might notice some possible side effects with the list of selected drugs. These possible interactions should ideally be displayed along with the drug list.

A *note* clause is provided in Vidwan to define such messages to be displayed along with the conclusion. This provides for a message to be attached with each rule in the rule base. The note feature is an optional one and may not be relevant for all domains.

Since each Vidwan rule has only one consequent, the note, in effect, allows a message to be attached with every consequent. At the time of displaying the conclusions, Vidwan checks if a note has been defined for each of the fired rules. If there is a note, it will be displayed under the heading "NOTES".

An example of a note for a rule about Ischaemic Heart Disease is given below:

```
note(a5, "IHD_attention: You show evidence of Ischaemic
        Heart Disease. Avoid all unusual physical effort.
        You need a Cardiologist's attention.").
```

where, a5 is the rule identifier of the corresponding rule. One can use "\n" to put line breaks in the notes when they are printed.

Vidwan provides a limited amount of run time formatting of notes. You can embed an attribute name inside the note string. When the note is to be displayed, the current value of the attribute is substituted in place of the name. For this, the attribute name should be enclosed inside a pair of "^" signs.

For example, a note such as "Your body temperature is ^temp^ which indicates high fever. You need to obtain immediate medical attention.", could be displayed as "Your body temperature is 102, which indicates high fever. You need to obtain immediate medical attention.", if the attribute *temp* has the value 102 during the consultation session. Where there are more than one value for the attribute, the value with the highest certainty factor will be used.

Vidwan allows users to display the conclusions of a session in an application dependent form using template files. In the default mode Vidwan's output of a session consists of the values obtained for the goal attribute and any notes applicable based on the set of rules fired. Use of a template file allows values of other attributes also to be displayed and other canned pieces of text to be inserted based on results of rules fired and values of various attributes deduced. We discuss this aspect in more detail later, in the section on report generator.

A.4 Antecedent of a Rule

The antecedents allow values of attributes to be compared against known values. An antecedent is a triple: attribute-operator-value, written in the form:

<opr> (<attr>, <value>)

The allowed operators are "is", "isnot", "notpos", "notneg" and "unknown" for non-numeric typed attributes. The meaning of these are as follows

is(A,V)	A is known to have value V
isnot(A,V)	A is known not to have value V
unknown (A,V)	It is unknown if A has value V (i.e., neither A is V nor A isnot V is known)
notpos(A,V)	either isnot(A,V) or unknown (A,V) (i.e., A is V is not true)
notneg(A,V)	either is (A,V) or unknown (A,V) (i.e., A is not V is not true)

In Vidwan a numeric figure, called the certainty factor, is associated with every attribute-value combination. This number represents the amount of belief the system has in the attribute having that particular value. This certainty factor lies in the range -1.0 to +1.0. The range of certainty factor for which the operators mentioned above are true are indicated in the following figure. Note that the range -0.2 to +0.2 are considered equivalent to zero. Figure A.1 shows the relations among these operators and the certainty factor in a pictorial form.

```
  |-------------------+-------+-------+--------------------|
-1.0                -0.2      0     +0.2                 +1.0

  <------------------->               <-------------------->
          isnot                                 is

                      <--------------->
                         unknown

  <----------------------------------->
                  notpos

                      <--------------------------------------->
                                           notneg
```

Figure A.1: Non-numeric Operators in Vidwan

For numeric attributes any of the following numeric comparison operators can be used: lt, gt, le, ge, eq and ne (standing respectively for less than, greater than, less than or equal to, greater than or equal to, equal to and not equal to).

The value field depends on the type of the attribute and the operator. For numeric attributes, the value should be a single number. If the attribute is boolean, the only value allowed is “true”. For boolean attributes note that checking if a value is false, is the same as checking if the value “isnot” true. For attributes of other types a string (consisting of alphabets and numeric characters only) is expected as the value. If the attribute has a menu defined, the value string should be present in the menu. Vidwan flags a warning otherwise.

Note that the value field cannot be another attribute name, irrespective of the type of the attribute.

Sometimes you may want to base a decision on whether an attribute has a value or not. For example, if the attribute disease could not get any value, it could indicate that the patient is alright. Or when you ask for temperature, the user may want to say unknown. To test for such situations, for all attributes, the value field accepts *unknown* as a valid value. For numeric attributes, if the value field is *unknown*, the operators *is* or *isnot* has to be used.

A.5 Consequent of a Rule

The consequent of a rule appears at the head of the rule:

rule(<id>, <attr>, <value>, <cf>) :- <anteset>.

In the above structure, the <attr>, <value> and <cf> fields make up the consequent in the rule. Consequents are used to set values to attributes.

The rule identifier <id> should be unique for every rule. The rule identifier is used by Vidwan to refer to rules during execution and also during explanation.

The attribute, <attr>, cannot be numeric typed. For numeric attributes, you should use the compute predicate which will be described in Section A.6.

Apart from the rule identifier <id>, the attribute <attr> and the value <value>, consequents also contain another field. This field is used to specify the certainty factor. The certainty factor <cf> used in the consequent indicates the certainty factor to be associated with the attribute-value combination, assuming all the antecedents are true with full certainty.

A.6 Compute

Vidwan is not expected to be used for number crunching operations. However, a few numerical operations are often required to be performed by a system like Vidwan. For instance, a rule base, which advises on cardiac diseases, might need to compare the weight of a patient with the height. It would be odd to ask a patient for his body mass index, when the system already knows the weight and height of the patient, since this could be computed using these values. A few numerical operations are provided in Vidwan to allow such simple calculations.

Attributes which have to be computed from other attributes or constants, can be defined using the compute predicate. This predicate has the form:

compute(<id>, <attr>, <expression>) :- <anteset>.

This means that the value of the attribute named <attr> is to be computed by evaluating the arithmetic expression <expression>, if the conditions defined by <anteset> hold.

The expression can contain $+$, $-$, $*$ and / as operators ($+$ for add, $-$ for subtract, $*$ for multiply and / for divide) and any constants or other numeric attributes as operands. The expression can also include the function min and max as in *min(income,30000)*0.3*.

The <id> is a unique identifier for the compute clause. In compute statements the <anteset> can be any conjunction of antecedents just as in a rule. The <anteset> can also be null (i.e., no antecedents), in which case the ":-" is omitted. The <attr> should be a numeric typed attribute. The value of an attribute may be defined using multiple compute clauses.

If a compute clause is defined for a numeric attribute, it will be used while evaluating the attribute; otherwise the user will be prompted for the value. A compute clause is applicable only if all the antecedents succeed. The value is computed using the first compute clause whose <anteset> evaluates to true. A null <anteset> is considered to be true. The user can thus define certain functions as follows:

compute(c1, attr1, (attr2 + attr3)) :- gt(attr2, 10.5).
compute(c2, attr1, (attr2 - attr3)) :- le(attr2, 10.5).

The second compute, c2, will be used if the first compute, c1, fails, i.e., if attr2 is not greater than 10.5.

A.7 Goal

In most cases, the goal attribute [1] for a given rule base is decided by the knowledge engineer. In such situations where the end user is not expected to know the name of the goal attribute, the goal clause can be used to set up the default goal attribute. The syntax of this clause is:

goal (<attr>).

If a goal attribute is defined, it is used as the default goal attribute during the consultation session. This can be over-ridden, if necessary, by giving a new attribute, when the system prompts for the name of the goal attribute.

The goal clause can be placed anywhere in the rule base. For style and layout reasons, it should be placed at the beginning of the rule base. If multiple goal definitions appear in a rule base, the one appearing last in the file will over-ride earlier definitions.

A.8 Comments

Comments can be inserted into a Vidwan rule base. If the first character in a line is "%" (the percentage sign), Vidwan treats that line as a comment and ignores that particular line, while loading the rule base. If a percent sign is encountered within a line, the rest of the line is treated as a comment and ignored.

[1] Typically, an expert system consultation session will be an attempt to identify possible values for an attribute. For a medical diagnosis expert system, the attribute may be disease. This attribute may be assigned values such as Rheumatic Heart Disease, Ischaemic Heart Disease, etc., at the end of the session. The attribute used in the rule base, which the system evaluates during a consultation session, is called the goal attribute.

A.9 Style and Layout

The type definitions should be placed at the beginning of the rule base. Ideally the layout of clauses should follow the following sequence:

1. Types
2. Templates, Menus and Notes
3. Computes and Rules

The goal clause can be placed anywhere in the rule base. However, users are advised to put it at the beginning of the rule base.

A.10 Report Generator in Vidwan

Vidwan offers a facility to generate detailed reports of a consultation session. This facility can also be used to control the layout and contents of the display of conclusions. To use this facility, a template file for the report should be created for the rule base. The system uses this template file to generate reports specific to a given run. The facility can be invoked by selecting the *rprt* option in the browse mode [2] of Vidwan.

Another template file is required for controlling the layout of the final result screen. However, the template file specification formats for both the files are the same. If the desired template file is not found, the system uses a default format for producing the final screen layout.

A.10.1 Using the Report Generator

To use the report generation facility for a rule base named xxx.rb, a template file named xxx.rpt should be present in the same directory as the rule base. For example, if you have a rule base named hruday.rb, the associated report template file should be named hruday.rpt. The template file for the final result screen should be named xxx.fin (in this example, hruday.fin).

The report for a consultation can be created by the user by choosing the option *rprt* in the browse mode. When the report facility is invoked the system checks for the required template file. If the template file is found, the user is asked to specify the file, where the report should be produced. The system processes the template file based on the information obtained in the current consultation and generates a report for it in the file specified.

A variety of options can be put in the template file to control the text to be included in the report. Values of attributes in the rule base can be accessed and incorporated. Text can be excluded or included based on various types of conditions. You can also include information obtained dynamically from the user, while creating the report. This can be used for incorporating details such as an address, license number etc. The next subsection describes the various options. An example template file and a sample output file obtained using that template file are also provided.

[2] The browse-mode is accessible at the end of a consultation session

A.10.2 The Template File

The template file essentially consists of free text. Some commands can be inserted in it, to control the text to be included in the output report, based on information obtained during the current consultation. *Each command should be given on a separate line and should start in the first column. The commands always start with a backslash.* The commands are divided into three classes:

- those incorporating values of certain attributes from the current session
- those including a certain piece of text only when certain conditions are true, (for example, if *disease* is found to be jaundice or if *age* is greater than 55)
- those controlling the layout of text.

Spaces in the template file are retained in the output file. These can be used for providing simple indentation. However, line breaks that should appear in the output should be explicitly specified using the commands mentioned below. Line breaks in the template file are ignored.

The various commands allowed in the template file are described below. The parts appearing in **bold** type are part of the command. The text in *italics* indicate generic fields such as attribute names, rule ids etc. For example, if you need to obtain the value(s) of an attribute you have to use the command **\all** *attributename*. If your attribute is named disease the actual command will be `\all disease`.

Accessing Values of Attributes

\value *attributename* This causes one value of the attribute to be output in the report at the current position. If the attribute has no value with CF above 0.2, the string `NOVALUE` is output. For example, if attribute disease has the value jaundice the command `\value disease` produces the output `jaundice` .

\valuec *attributename* This causes one value of the attribute to be output in the report at the current position, along with the associated certainty factor. If the attribute has no value (with CF above 0.2), the string `NOVALUE` is output. The certainty factor is enclosed in normal brackets. For example, if attribute disease has the value jaundice with a certainty factor of 0.6, the command `\valuec disease` produces the output `jaundice (0.6)`.

\all *attributename* All values currently available for the attribute are output in the report. The values are separated by commas and connected by an "and" at the end. Note that only values with CF above 0.2 are considered. For example, if the values of an attribute named complaint are chest-pain, swelling-of-feet and tiredness, the command `\all complaint` will produce the output `chest-pain, swelling-of-feet and tiredness`.

allc *attributename* All values currently available for the attribute are output in the report, along with the associated certainty factors. The values are separated by commas and connected by an "and" at the end. The certainty factors appear in brackets right after the value. Note that, only values with CF above 0.2 are printed. For example, if the values of an attribute named complaint are chest-pain with CF = 0.6, swelling-of-feet with CF = 0.5 and tiredness with CF = 0.1, the command `\allc complaint` will produce the output `chest-pain (0.6) and swelling-of-feet (0.5)`.

cf *attributename value* The current certainty factor associated with the attribute-value combination is output at the current position. For example, if attribute complaint has value chest-pain with CF = 0.6, the command `\cf complaint chest-pain` produces the output `0.6`. If the particular value is not known for the attribute a value of 0.0 is output.

ask *text* All the text appearing after the word *ask,* upto the end of the current line, is output to the user on the terminal at this point. The system then waits for the user to enter a line of text in response. The text entered is inserted into the output report at the current position. This option can be used to get details such as licence numbers, addresses, etc. which may be required in the report, but may not be relevant to the expert system. For example, if you would like to insert the address of the user in the report file, you would use the command `\ask "Please enter your address"`.

response *attributename* This option outputs the current values and associated CFs, of the attribute in a format used in the response files of Vidwan. Please see the Vidwan user manual [NCST, 1993] for the syntax of response files. This option is useful when you want to run multiple rule bases in sequence and the values deduced in one forms part of the input to the subsequent one(s). At the end of running the earlier rule base, you can use the report or result screen option, with a template file containing only this command for all relevant attributes, to create a file in the response file format. For the subsequent rule bases, this can be specified as a response file to be loaded. Also see the Section on embedding of Vidwan, in this chapter.

In all these commands, if the attribute name is unknown, an error message is displayed to the user and the command ignored.

Some of these options allow the user to specify some formatting directives indicating the number of spaces to be used to display the values.

Selective Inclusion of Text

The commands in this section allow you to include a piece of text if certain conditions are true. The end of the relevant piece of text should be indicated by the command **end**. The commands can be nested to arbitrary depth. Each command in this section requires a corresponding **end** after the text covered by the command. Note that the text covered by such a command can also contain other formatting commands.

\iffired *ruleid ruleid ruleid ...* If any of the named rules has fired in the current run, the text from the next line onwards till the corresponding "\end" command is processed. Otherwise the text is ignored.

\ifbound *attributename* Process the following text (upto its corresponding "\end") only if the named attribute has obtained some value in the current run. An attribute is said to have a value if it has a value with CF above 0.2. For boolean attributes, a CF below -0.2 is also adequate for the attribute to have a value.

\ifubound *attributename* Process the following text (upto its corresponding "\end") only if the named attribute has failed to obtain any value in the current run. An attribute is said to have a value if it has a value with CF above 0.2. For boolean attributes, a cf below -0.2 is also adequate for the attribute to have a value. In all other case, the attribute is treated as unbound.

\if *operator attributename value* Process the following text (upto its corresponding "\end") only if the condition, viz, attribute opr value, viewed as an antecedent in the rule base evaluates to true, in the current consultation. Any combination/condition that is allowed as an antecedent in a rule is allowed here. Note that the syntax is however different. Thus, to output a message if the patient complains of tiredness, you would use the following:

```
\if is complaint tiredness
 ... the message to be included ...
\end
```

Controlling Layout

No major formatting support is provided in the current version of Vidwan. Vidwan, however, allows the following:

Comments Any line starting with a % sign is recognised as a comment and ignored by the system.

Line breaks The command **\line** forces a line break in the output at the current point in the report.

Paragraph breaks The command **\par** provides a blank line to start a new paragraph at the current point in the report.

Controlling length of fields The options \value, \valuec, \all and \allc can be given optional format specifications to restrict the number of characters in the output. For example, \value num1 6.2 will display the value of the numerical attribute num1 using 6 character spaces with 2 spaces after the decimal point. If the value were 13.63, it will appear as `bb13.63` where letter "b" here denotes a blank space. In general for "f" type attributes the total number of spaces to be used and the number of spaces after decimal point can be specified, as shown in the example. For "i" type attributes, only a single number representing the total number of spaces can be specified. For other types (whose values are character strings) also, the total number of spaces to be used can be specified.

A.11 Text Database Access Facility

Vidwan can access values of attributes from databases maintained as simple text files. This facility provides a short cut which avoids having to write separate programs (see previous section) for interfacing to such databases. The database is assumed to consist of a set of records, with each record on a separate line. The field separator is assumed to be a colon or a blank. Lines starting with a % in such databases are assumed to be comments, and not processed.

The specification of database access in Vidwan rule base has the following form:

textdb(*attribute,database name,key,field number***).**

- The attribute refers to the attribute whose value is to be obtained from the database.
- The database name is the name of the file to be used as the database. The full name should be specified. If the file is not in the current directory, the full path name must be specified. The name must be enclosed in quotes.
- The key is the name of the attribute which should be used as the key in accessing the right record in the database. This attribute should have a value before the database call is invoked. It is assumed that the first field in the record will be used as the key. The key specification only indicates the attribute whose value should be used for matching with this field to identify the correct record. Attributes having boolean or floating point type, cannot be used as the key attribute.
- The field number indicates the field whose value should be retrieved as the value of the desired attribute.

Example

Consider a database SALINFO.DB giving salary information about various employees in an organisation in the form:

IDCODE NAME BASICPAY

Assume that Vidwan attributes named idcode and basic_pay exist in the rule base. The basic_pay is to be accessed from the database using idcode as the key. The idcode is evaluated in the initial stages of consultation. The database access specification will be as follows:

textdb(basic_pay, 'SALINFO.DB', idcode, 3).

The number 3 indicates that basic pay is the 3rd field in the database record.

A.12 Embedding of Vidwan

Vidwan can be embedded in other programs by using a command file to control Vidwan execution.

A.12.1 Command File based Invocation of Vidwan

A command file for Vidwan is a text file containing one or more of the following commands. The commands are executed in the order specified in the command file.

1. load *rulebasefile*
2. response *responsefile*
3. run [*goalattribute*]
4. printresults *resultfile*
5. makereport *reportfile*
6. saveresponse *responsefile*
7. browsemode

The words in *italics* indicate slots to be filled in depending on the application. The goal attribute is optional for the run command and hence shown in square brackets. "run" without any argument uses the goal attribute defined in the rule base as the goal.

The commands, if present, should be in the order given above. However, none of these commands are compulsory.

If a command file is specified as a command line argument, Vidwan executes the commands specified in the command file. At the end of the command sequence, Vidwan terminates execution. During the run, if there are attributes not obtained from the response file, the user will be asked for their values. Similarly, browsemode command will also cause Vidwan screens to be displayed. In all other cases, the use of Vidwan is normally not visible to the user.

A.12.2 Embedding Vidwan using Command File

An application requiring the use of an expert system module in its process (where the expert system module is implemented in Vidwan), can invoke Vidwan within the program by setting up an appropriate command file[3]. If most data required for the expert system are available in the program, a response file may be set up and specified in the command file. The communication of results back to the program will have to be done by generating a file using the printresults or makereport commands in the command file. Typically

[3]See Chapter 13 for an example

the caller would expect the results in a terse format to make its parsing task easy. And if a detailed report of the session is required, the "printresults" option can be used with an appropriate template file for the communication of the required values back to the program and the makereport option may be used to obtain the detailed report of the expert system session.

Invocation with Command File

```
vidwan -cmycomfile
```

uses "mycomfile" as the command file. There should be no blank between c and the filename.

B. Hruday: Rule Base and Sample Run

B.1 Hruday Rule Base

```
%HRUDAY V 2: Heart Diseases Advisor
%
% (c) 1993. NCST Bombay
%
% Hruday is a demonstration ES. It offers the ‘common man’ advice
% about heart problems (physical ones!). We advise readers to remember
% that it is demo expert system, and they should not take its advice
% as a substitute for medical opinion.

% Hruday simulates a friend who is connected with the medical
% profession, not necessarily as a cardiologist, and as one who offers
% information, and safe advice. Hruday does not offer to make a firm
% diagnosis, nor does it prescribe treatment. It leaves all this to a
% specialist. The idea is to familiarise the user with the issues
% concerning heart diseases, and make him understand his situation
% better.

% Look under Template Definitions for an alphabetized list
% of questions. This list can also be used as a glossary of
% terms employed, while reading the rule-base. Some parameters
% do not really need templates, because the user is never asked
% questions about them. They may, however, be associated with
% templates to enable the user to find out about them easily.
% Any time Vidwan asks you for an attribute-name, you can invoke
% help, and type in words associated with that attribute. Vidwan
% uses the template definitions to present you with a set of
% relevant attribute and their definitions. The templates used for
% this purpose can easily be identified, as they do not have any
% question mark in the definition part.
```

Type Definitions

```
type(adv, m).
type(disease, m).
type(suspected_disease, m).
type(sweating_linked_cp, b).
type(card_symp_present, b).
```

```
type(chest_pain_type, m).
type(diabetic, p).
type(complaint, m).
type(chest_pain_locn, m).
type(managerial, p).
type(sedentary, p).
type(sex, p).
type(bp_status, p).
type(number_of_drinks, f).
type(number_of_cigs, f).
type(smoker, b).
type(pregnant, p).
type(sore_throat_fever, b).
type(PND, b).
type(age, f).
type(cp_duration, p).
type(nails_blue, b).
type(fever_with_jntpain, b).
type(kidney_disease, b).
type(filariasis, b).
type(rfever_evidence, b).
type(hypertensive, b).
type(IH_risk, b).
type(hb_level, f).
type(systolic_bp, f).
type(diastolic_bp, f).
type(palpitations, b).
type(childhood_rfever, b).
type(alcoholic, b).
type(lung_disease, b).
type(chronic_penicil_usr, b).
type(breath_linked_cp, b).
type(ambitious, b).
type(orthopnoea, b).
type(cough_linked_cp, b).
type(exertion_linked_cp, b).
type(cp_radiates, b).
type(bypass_surg, b).
type(val_surgery, b).
type(HD_confirmation, b).
type(face_swelling, b).
type(chorea, b).
type(clubbed_fingers, b).
type(dysp_at_rest, b).
type(dysp_on_exertion, b).
type(epileptic, b).
type(bmi, f).
type(weight, f).
type(height, f).
```

Goal Definition

```
goal(adv).
```

Template Definitions

```
template(adv, "Advice to patient ...").
template(age, "What is your age (in years)?").
template(ambitious, "Do you think you are an ambitious person, who
        pushes \n oneself hard?").
template(bp_status, "Do you know your blood pressure status?").
template(breath_linked_cp, "Does the chest pain become worse on
        deep\n breathing, lying down or on pressing at the site of
        pain?").
template(bypass_surg, "Have you undergone any bypass surgery?").
template(card_symp_present, "cardiac symptoms are present").
template(chest_pain_type, "typeof the chest pain").
template(chest_pain_locn, "Where do you feel the chest pain?").
template(childhood_rfever, "Did you suffer from rheumatic fever
        in \n childhood?").
template(complaint, "Do you have one or more of the following
        complaints?").
template(cp_duration, "How long does your chest pain last?").
template(chorea, "Did you suffer from abnormal limb movements in
        your \n childhood?").
template(chronic_penicil_usr, "Were you ever advised to take
        penicillin on \n a long-term basis?").
template(clubbed_fingers, "Are your finger tips broadened, showing
        unusual \n convexity?").
template(cough_linked_cp, "Is your chest pain felt mainly during
        coughs?").
template(cp_radiates, "Does the chest pain radiate to other parts
        of the \n body: left arm, jaw, or back?").
template(diabetic, "Are you a diabetic?").
template(diastolic_bp, "What is your diastolic blood pressure?").
template(disease, "disease identified").
template(dysp_at_rest, "Do you have breathlessness even at rest,
        sometimes?").
template(dysp_on_exertion, "Do you have breathlessness even while
        performing \n tasks you are accustomed to?").
template(epileptic, "Do you suffer from epilepsy (fits)?").
template(exertion_linked_cp, "Does your chest pain become worse on
        exertion?").
template(face_swelling, "Was there any swelling of face before the
        swelling \n of feet?").
template(fever_with_jntpain, "Did you suffer from fever accompanied
        by \n joint pains?").
template(filariasis, "Have you ever been told that you have
        filariasis?").
template(hb_level, "What is the value of your blood haemoglobin?").
template(HD_confirmation, "cardiac factors are definitely
        involved").
template(height, "What is your height (in cm)?").
template(hypertensive, "patient has blood pressure (hypertension)").
template(IH_risk, "the patient has risk of ischaemic heart
        disease").
template(kidney_disease, "Have you ever been told that you have a
        kidney \n problem?").
template(lung_disease, "Are you a chronic patient of lung disease
        such \n as Asthma?").
```

```
template(managerial, "Is the type of your job managerial in
        nature?").
template(nails_blue, "Do you have a bluish tinge on your finger
        nails?").
template(number_of_cigs, "How many cigarettes do you smoke per day,
        on \n the average?").
template(number_of_drinks, "How many drinks do you have per day, on
        the \n average? Count anything with 30 ml of alcohol as
        one drink. Count \n each glass of beer as one drink.").
template(orthopnoea, "Do you feel more comfortable in an upright
        position \n than when you are lying down?").
template(palpitations, "Do you sense your heart beating prominently
        even \n when you are not taxing yourself physically?").
template(PND, "Do you often wake up from sleep feeling breathless
        or choked?").
template(pregnant, "Are you pregnant?").
template(rfever_evidence, "patient shows evidence of rheumatic
        fever").
template(sedentary, "Is the type of your job sedentary in nature?").
template(sex, "Are you male or female?").
template(sore_throat_fever, "Have you had sore throat followed by
        fever \n with joint pain? ").
template(sweating_linked_cp, "Is there any sweating or giddiness
        associated \n with your chest pain? ").
template(suspected_disease, "the disease suspected").
template(systolic_bp, "What is your systolic blood pressure?").
template(val_surgery, "Have you undergone any valvular surgery?").
template(weight, "What is your weight (in Kg)?").
```

Menu Definitions

```
menu(diabetic, [yes, no, unknown]).
menu(complaint, [swelling_of_feet, tiredness, syncope, chest_pain]).
menu(chest_pain_locn, [left_side, center_of_chest, right_side]).
menu(managerial, [yes, no]).
menu(sedentary, [yes, no]).
menu(sex, [male, female]).
menu(bp_status, [yes, no]).
menu(pregnant, [yes, no]).
menu(age, [0.0,120.0]).
menu(cp_duration, [a_few_minutes,half_hour,few_hours]).
menu(hb_level, [1.0,30.0]).
menu(systolic_bp, [100.0,210.0]).
menu(diastolic_bp, [30.0,200.0]).
```

Compute Definitions

```
compute (c1, bmi, (weight * 10000.0)/(height * height)).
```

The Rules

Rules for adv

```
rule(a3, adv, diet, 0.90) :-
     is(HD_confirmation, true),
     gt(bmi, 30.0).
rule(a301, adv, diet, 0.82) :-
     is(HD_confirmation, true),
     gt(bmi, 24.0),
     le(bmi, 30.0).
rule(a2, adv, hbp, 1.00) :-
     is(hypertensive, true).
rule(a4, adv, strain, 0.95) :-
     is(disease, IHD),
     is(sedentary, yes),
     is(managerial, yes).
rule(a401, adv, strain, 0.80) :-
     is(disease, IHD),
     is(sedentary, no),
     is(managerial, yes).
rule(a5, adv, IHD_attn, 1.00) :-
     is(disease, IHD).
rule(a602, adv, dont_smoke, 0.75) :-
     isnot(disease, IHD),
     is(hypertensive, true),
     ge(number_of_cigs, 5.0).
rule(a601, adv, dont_smoke, 0.80) :-
     is(disease, IHD),
     ge(number_of_cigs, 5.0),
     le(number_of_cigs, 10.0).
rule(a6, adv, dont_smoke, 0.85) :-
     is(disease, IHD),
     gt(number_of_cigs, 10.0).
rule(a7, adv, RHD_attn, 1.00) :-
     is(disease, RHD).
rule(a8, adv, antibiotics, 1.00) :-
     is(disease, RHD),
     isnot(childhood_rfever, true).
rule(a9, adv, CHD_attn, 1.00) :-
     is(disease, CHD).
rule(a10, adv, less_salt, 0.80) :-
     is(HD_confirmation, true),
     is(hypertensive, true).
rule(a11, adv, diabetes_attn, 1.00) :-
     is(HD_confirmation, true),
     is(diabetic, yes).
rule(a12, adv, CARP, 1.00) :-
     is(disease, CARP).
rule(a13, adv, no_evid, 1.00) :-
     is(card_symp_present, true),
     isnot(HD_confirmation, true),
     isnot(disease, RHD),
     isnot(disease, CHD),
     isnot(disease, IHD),
     isnot(disease, CARP).
```

```
rule(a14, adv, watch, 1.00) :-
     is(card_symp_present, true),
     is(IH_risk, true),
     isnot(HD_confirmation, true).
rule(a15, adv, special, 1.00) :-
     is(card_symp_present, true),
     is(HD_confirmation, true),
     isnot(disease, IHD),
     isnot(disease, RHD),
     isnot(disease, CHD),
     isnot(disease, CARP).
rule(a16, adv, quit_alcohol, 1.00) :-
     is(disease, CARP),
     is(alcoholic, true).
```

Rules for card_symp_present

```
rule(cs3, card_symp_present, true, 0.30) :-
     is(complaint, swelling_of_feet).
rule(cs1, card_symp_present, true, 0.40) :-
     is(dysp_on_exertion, true).
rule(cs2, card_symp_present, true, 0.40) :-
     is(palpitations, true).
rule(cs4, card_symp_present, true, 0.08) :-
     is(complaint, tiredness).
rule(cs5, card_symp_present, true, 0.40) :-
     is(nails_blue, true).
rule(cs6, card_symp_present, true, 0.18) :-
     is(complaint, syncope).
rule(cs8, card_symp_present, true, 0.40) :-
     is(complaint, chest_pain).
```

Rules for chest_pain_type

```
rule(ch1, chest_pain_type, cardiac, 0.25) :-
     is(complaint, chest_pain),
     is(chest_pain_locn, center_of_chest).
rule(ch2, chest_pain_type, cardiac, 0.25) :-
     is(complaint, chest_pain),
     is(chest_pain_locn, left_side).
rule(ch3, chest_pain_type, cardiac, -0.18) :-
     is(complaint, chest_pain),
     is(chest_pain_locn, right_side).
rule(ch5, chest_pain_type, cardiac, 0.35) :-
     is(complaint, chest_pain),
     is(cp_radiates, true).
rule(ch6, chest_pain_type, cardiac, -0.18) :-
     is(complaint, chest_pain),
     is(cough_linked_cp, true).
rule(ch7, chest_pain_type, cardiac, 0.40) :-
     is(complaint, chest_pain),
     is(sweating_linked_cp, true).
rule(ch8, chest_pain_type, cardiac, 0.40) :-
     is(complaint, chest_pain),
     is(exertion_linked_cp, true).
```

```
rule(ch9, chest_pain_type, cardiac, -0.20) :-
     is(complaint, chest_pain),
     is(breath_linked_cp, true).
rule(ch10, chest_pain_type, cardiac, -0.20) :-
     is(complaint, chest_pain),
     is(lung_disease, true).
rule(ch11, chest_pain_type, cardiac, 0.10) :-
     is(complaint, chest_pain),
     is(cp_duration, few_hours).
rule(ch12, chest_pain_type, cardiac, 0.25) :-
     is(complaint, chest_pain),
     is(cp_duration, half_hour).
rule(ch4, chest_pain_type, cardiac, 0.10) :-
     is(complaint, chest_pain),
     is(cp_duration, a_few_minutes).
```

Rules for suspected_disease

```
rule(sd1, suspected_disease, IHD, 0.60) :-
     is(card_symp_present, true),
     is(IH_risk, true).
rule(sd2, suspected_disease, IHD, 0.18) :-
     is(cp_radiates, true).
rule(sd3, suspected_disease, RHD, 0.18) :-
     is(card_symp_present, true),
     gt(age, 10.0),
     le(age, 40.0).
rule(sd4, suspected_disease, RHD, 0.80) :-
     is(card_symp_present, true),
     is(HD_confirmation, true),
     isnot(IH_risk, true),
     is(val_surgery, true).
rule(sd5, suspected_disease, CHD, 0.80) :-
     is(card_symp_present, true),
     lt(age, 40.0),
     is(nails_blue, true).
rule(sd6, suspected_disease, CHD, 0.18) :-
     is(card_symp_present, true),
     lt(age, 40.0).
rule(sd7, suspected_disease, CHD, 0.18) :-
     is(card_symp_present, true),
     lt(age, 15.0).
```

Rules for IH_risk

```
rule(ihr2, IH_risk, true, 0.15) :-
     is(sedentary, yes).
rule(ihr3, IH_risk, true, 0.10) :-
     is(managerial, yes).
rule(ihr4, IH_risk, true, 0.15) :-
     is(ambitious, true).
rule(ihr5, IH_risk, true, 0.20) :-
     is(smoker, true).
rule(ihr6, IH_risk, true, 0.20) :-
     is(hypertensive, true).
```

```
rule(ihr7, IH_risk, true, 0.10) :-
     is(sex, male).
rule(ihr9, IH_risk, true, 0.10) :-
     gt(age, 30.0).
rule(ihr10, IH_risk, true, 0.10) :-
     gt(age, 50.0).
rule(ihr11, IH_risk, true, 0.10) :-
     is(sex, female),
     gt(age, 55.0).
rule(ihr12, IH_risk, true, 0.20) :-
     gt(bmi, 24.0).
```

Rules for rfever_evidence

```
rule(rf1, rfever_evidence, true, 0.60) :-
     isnot(childhood_rfever, true),
     is(chronic_penicil_usr, true).
rule(rf2, rfever_evidence, true, 0.40) :-
     isnot(childhood_rfever, true),
     is(fever_with_jntpain, true).
rule(rf3, rfever_evidence, true, 0.21) :-
     isnot(childhood_rfever, true),
     is(fever_with_jntpain, true),
     is(sore_throat_fever, true).
```

Rules for disease

```
rule(dis1, disease, IHD, 1.00) :-
     is(suspected_disease, IHD),
     is(HD_confirmation, true).
rule(dis2, disease, IHD, 1.00) :-
     is(suspected_disease, IHD),
     isnot(IH_risk, true),
     gt(age, 40.0),
     is(bypass_surg, true).
rule(dis3, disease, RHD, 1.00) :-
     is(suspected_disease, RHD),
     is(HD_confirmation, true).
rule(dis4, disease, RHD, 0.40) :-
     is(suspected_disease, RHD),
     is(childhood_rfever, true).
rule(dis5, disease, RHD, 0.18) :-
     is(suspected_disease, RHD),
     isnot(HD_confirmation, true).
rule(dis6, disease, RHD, 0.60) :-
     is(suspected_disease, RHD),
     is(rfever_evidence, true).
rule(dis7, disease, RHD, 0.40) :-
     is(suspected_disease, RHD),
     is(chorea, true).
rule(dis8, disease, CHD, 0.60) :-
     lt(age, 15.0),
     is(HD_confirmation, true),
     gt(hb_level, 15.0).
rule(dis9, disease, CHD, 0.60) :-
```

```
     lt(age, 15.0),
     is(HD_confirmation, true),
     is(nails_blue, true).
rule(dis10, disease, CHD, 0.35) :-
     is(suspected_disease, CHD),
     is(clubbed_fingers, true).
rule(dis11, disease, CHD, -0.21) :-
     is(nails_blue, true),
     gt(age, 40.0),
     is(lung_disease, true).
rule(dis12, disease, CARP, 0.45) :-
     gt(age, 40.0),
     is(HD_confirmation, true),
     is(nails_blue, true).
rule(dis13, disease, CHD, -0.11) :-
     is(nails_blue, true),
     gt(age, 40.0),
     is(smoker, true).
```

Rules for HD_confirmation

```
rule(cf1, HD_confirmation, true, 0.75) :-
     is(card_symp_present, true),
     is(chest_pain_type, cardiac).
rule(cf2, HD_confirmation, true, -0.20) :-
     is(complaint, syncope),
     is(epileptic, true).
rule(cf3, HD_confirmation, true, -0.20) :-
     is(complaint, syncope),
     is(diabetic, yes).
rule(cf4, HD_confirmation, true, -0.40) :-
     is(complaint, syncope),
     is(hypertensive, true).
rule(cf5, HD_confirmation, true, 0.60) :-
     is(dysp_on_exertion, true).
rule(cf6, HD_confirmation, true, 0.60) :-
     is(dysp_on_exertion, true),
     is(dysp_at_rest, true).

rule(cf7, HD_confirmation, true, 0.60) :-
     is(dysp_on_exertion, true),
     is(PND, true).
rule(cf8, HD_confirmation, true, -0.40) :-
     is(dysp_on_exertion, true),
     is(lung_disease, true).
rule(cf9, HD_confirmation, true, 0.40) :-
     is(palpitations, true).
rule(cf10, HD_confirmation, true, 0.20) :-
     is(complaint, swelling_of_feet).
rule(cf11, HD_confirmation, true, 0.20) :-
     is(complaint, swelling_of_feet),
     is(sex, female),
     gt(age, 15.0),
     lt(age, 45.0),
     isnot(pregnant, yes).
```

```
rule(cf12, HD_confirmation, true, -0.40) :-
     is(complaint, swelling_of_feet),
     is(kidney_disease, true).
rule(cf13, HD_confirmation, true, -0.40) :-
     is(complaint, swelling_of_feet),
     is(face_swelling, true).
rule(cf14, HD_confirmation, true, 0.20) :-
     is(dysp_on_exertion, true),
     is(orthopnoea, true).
rule(cf15, HD_confirmation, true, -0.40) :-
     is(PND, true),
     is(lung_disease, true).
rule(cf16, HD_confirmation, true, -0.40) :-
     is(complaint, swelling_of_feet),
     is(filariasis, true).
```

Other Rules

```
rule(ht1, hypertensive, true, 1.00) :-
     is(bp_status, yes),
     ge(systolic_bp, 160.0).
rule(ht2, hypertensive, true, 1.00) :-
     is(bp_status, yes),
     ge(diastolic_bp, 95.0).
rule(cig1, smoker, true, 1.00) :-
     gt(number_of_cigs, 19.0).
rule(cig2, smoker, true, 0.60) :-
     gt(number_of_cigs, 9.0).
rule(cig3, smoker, true, 0.30) :-
     gt(number_of_cigs, 4.0).
rule(cig4, smoker, true, 0.10) :-
     le(number_of_cigs, 4.0),
     gt(number_of_cigs, 0.0).
rule(cig5, smoker, true, -1.00) :-
     le(number_of_cigs, 0.0).
rule(alco1, alcoholic, true, 1.00) :-
     gt(number_of_drinks, 2.0).
rule(alco2, alcoholic, true, 0.60) :-
     gt(number_of_drinks, 1.0).
rule(alco3, alcoholic, true, 0.30) :-
     gt(number_of_drinks, 0.5).
rule(alco4, alcoholic, true, -1.00) :-
     le(number_of_drinks, 0.5).
```

Final Result Display Format

```
% This is the file hruday.fin used to specify the format for
% displaying the conclusions arrived at after the consultation.
% Vidwan uses the report generator facility described in the appendix
% A of this book for creating the output based on this file.

\par
\iffired a5
```

```
You show evidence of ischaemic heart disease. Avoid all unusual
physical effort. You need a cardiologist’s attention. (
\cf adv IHD_attn
)
\end
\iffired a7
\par
You are probably suffering from rheumatic heart disease.  Consult a
cardiologist. (
\cf adv RHD_attn
)
\end
\iffired a9
\par
You seem to have some kind of congenital heart disease. This is
usually curable by surgery. Consult a cardiologist and get further
investigation done. (
\cf adv CHD_attn
)
\end
\iffired a12
\par
You seem to have a type of heart disease called Cor Pulmonale.
Smoking and drinking aggravate it. (
\cf adv CARP
)
\end
\iffired a13
\par
What you have said does not indicate the presence of heart disease.
You seem to be alright. But a periodical medical checkup is always
advisable. (
\cf adv no_evid
)
\end

\iffired a14
\par
You run some risk of developing heart disease, because of your
life-style. But, at present you seem to be alright.  Make sure you
undergo periodical medical checkups. (
\cf adv watch
)
\end
\iffired a15
\par
You should consult a heart specialist. You show some signs of heart
disease which need further investigation. (
\cf adv special
)
\end
\iffired a8
\par
You might need to take antibiotics, probably penicillin, when you
have fever. This is to prevent Rheumatic fever causing any damage to
the heart. Consult a physician about this. (
```

```
\cf adv antibiotics
)
\end
\iffired a11
\par
You have to have regular treatment for your diabetes. (
\cf adv diabetes_attn
)
\end
\iffired a16
You should avoid heavy alcohol intake. (
\cf adv quit_alcohol
)
\end
\iffired a2
Watch your blood pressure, and keep it under control. (
\cf adv hbp
)
\end
\iffired a10
Take less salt in your food. (
\cf adv a10
)
\end
\iffired a602
\par
Give up smoking, if you can. It worsens your blood pressure. (
\cf adv dont_smoke
)
\end



\iffired a601
\par
Give up smoking, if you can. (
\cf adv dont_smoke
)
\end
\iffired a6
\par
Give up smoking. At the least, make a drastic reduction. That will
help a lot. (
\cf adv dont_smoke
)
\end
\iffired a3
\par
You have to go on a diet and reduce your weight.  Cut down high
cholesterol foods like dairy products and fried items. (
\cf adv diet
)
\end
\iffired a301
\par
```

```
Try to reduce your weight and cut down high cholesterol foods like
dairy products and fried items. (
\cf adv diet
)
\end
\iffired a4 a401
\par
Avoid mental strain and tension where possible. (
\cf adv strain
)
\end
```

B.2 A Sample Run

The following is the log of a consultation session with Hruday. The text in *this font are annotations on the log.* The user responses are shown in **bold**.

Enter goal attribute (quit: main menu; ¡CR¿: default=adv):
Enter the response filename(¡CR¿ if none):

Do you have one or more of the following complaints?
>> **swelling_of_feet**
CF= 1.00
>> **chest_pain**
CF= 1.00

Do you have breathlessness even while performing tasks you are accustomed to?
>> **0.60**

Do you sense your heart beating prominently even when you are not taxing yourself physically?
>> **1.00**

Do you have a bluish tinge on your finger nails?
>> **-1.00**

Where do you feel the chest pain?
>> **left_side**
CF= 1.00 >> **center_of_chest CF= 0.30**

Does the chest pain radiate to other parts of the body: left arm, jaw, or back?
>> **1.00**

Is your chest pain felt mainly during coughs?
>> **-0.60**

Is there any sweating or giddiness associated with your chest pain?
>> **1.00**

Does your chest pain become worse on exertion?
>> **0.60**

Does the chest pain become worse on deep breathing, lying down or on pressing at the site of pain?
>> **-0.30**

Are you a chronic patient of lung disease such as Asthma?
>> **-1.00**

How long does your chest pain last?
>>**half_hour**

Do you have breathlessness even at rest, sometimes?
>> **-0.30**

Do you often wake up from sleep feeling breathless or choked?
>> **0.30**

What is your age (in years)?
>> **45**

Are you male or female?
>> **male**

Have you ever been told that you have a kidney problem?
>> **-0.60**

Was there any swelling of face before the swelling of feet?
>> **-0.60**

Do you feel more comfortable in an upright position than when you are lying down?
>> **-0.60**

Have you ever been told that you have filariasis?
>> **-0.60**

What is your weight (in Kg)?
>>**87**

What is your height (in cm)?
>> **173**

Do you know your blood pressure status?
>> **yes**

What is your systolic blood pressure?
>> **128**

What is your diastolic blood pressure?
>>**90**

Is the type of your job sedentary in nature?
>> **yes**

Is the type of your job managerial in nature?
>> **yes**

Do you think you are an ambitious person, who pushes oneself hard?
>> **1.00**

How many cigarettes do you smoke per day, on the average?
>> **12**

Are you a diabetic?
>>**no**

The interaction ends here. The report generated by Vidwan *using the template in Section B.1 is given below.*

You show evidence of ischaemic heart disease. Avoid all unusual physical ef fort. You need a cardiologist's attention. (0.51)

Give up smoking. At the least, make a drastic reduction. That will help a lot. (0.44)

Try to reduce your weight and cut down high cholesterol foods like dairy products and fried items. (0.74)

Avoid mental strain and tension where possible. (0.49)

Enter your option (resp/help/how/shrl/shvl/rslt/quit/rprt/prnt/debg/w-if): **resp**
What is the response filename: **alfa**
Responses saved in the file alfa.res

C. GoodShow: Rule Base and Sample Run

C.1 GoodShow Rule Base

Type Definitions

```
%% 0-level askable attributes. Values of these attributes
%%  are obtained from the user.

type(problem_with_picture, b).
type(problem_with_sound, b).
type(picture_seen_on_screen, p).
type(type_of_picture_problem, m).
type(TV_age, i).
type(dark_screen, p).
type(complete_silence, p).
type(snow_on_screen, b).
type(panel_lights, p).
type(picture_size_abnormal, b).
type(brightness_too_low, b).
type(contrast_too_high, b).
type(type_of_raster, p).
type(picture_misaligned, b).
type(height_problem, b).
type(width_problem, b).
type(colour_seen_on_screen, p).
type(type_of_colour_problem, m).
type(Lines_Dots_Netting_seen, b).
type(Ghosts_seen, b).
type(horizontal_shift, b).
type(vertical_shift, b).
type(unsteadiness, p).
type(rolling_direction, p).
type(type_of_jumping, p).
type(sound_distorts_with_picture, b).
type(plug_point_supply, b).

%% 1-level derivable attributes.

type(TV_dead, b).
type(snowy_picture, b).
type(size_problem, b).
type(shift_problem, b).
```

```
type(suspect_sound_ckt, b).
type(suspect_sync_ckt, b).

%% 2-level derivable attributes

type(raster_present, b).
type(raster_absent, b).
type(sound_present, b).
type(sound_absent, b).
type(contrast_problem, b).
type(brightness_problem, b).
type(dimension_problem, b).
type(colour_problem, b).
type(rolling_problem, b).
type(jumping_problem, b).
type(suspect_luma_ckt, b).
type(suspect_picture_tube, b).
type(suspect_colour_ckt, b).
type(suspect_IF_amp, b).
type(suspect_ante_tuner, b).

%% 3-level derivable attributes

type(suspect_Antenna_or_Tuner, b).

%% 4-level derivable attributes

type(faulty_module, m).
```

Goal Definition

```
goal(faulty_module).
```

Template Definitions

```
template(faulty_module, "The following modules are probably faulty in
         your TV set:\n\n").
template(problem_with_picture, "Does the TV have any problem with the
         picture?").
template(problem_with_sound, "Does the TV have any problem with
         the sound?").
template(picture_seen_on_screen, "Do you see any picture at all on
         the screen?").
template(type_of_picture_problem, "Which of the following display
         characteristics show a problem?").
template(TV_age, "How old is your TV set (in years)?").
template(dark_screen, "Does the screen of TV look completely
         black?").
template(complete_silence, "Is your TV completely silent -- not
         producing any sound at all?").
template(snow_on_screen,"Is the screen full of tiny coloured
         spots?").
template(panel_lights, "What is the status of the front
         panel lights?").
```

```
template(picture_size_abnormal, "Do you feel that the size of the
         pictures on the screen is abnormal?").
template(brightness_too_low, "Do you feel that the brightness of the
         picture is too low?").
template(contrast_too_high, "Is the contrast of the picture
         too high?").
template(type_of_raster, "Which of the following types of raster
         scans do you see on the screen?").
template(picture_misaligned, "Does the picture look misaligned on the
         screen?").
template(height_problem, "Is the picture distorted vertically?\n For
         example, do people appear taller or shorter than they
         are?").
template(width_problem, "Is the picture distorted horizontally?\n For
         example, do people appear thinner or fatter than they
         are?").
template(Lines_Dots_Netting_seen, "Do you see stray lines, dots or
         netting patterns on the screen?").
template(Ghosts_seen, "Do you see ghost images (multiple shadows of
         the pictures) on the screen?").
template(horizontal_shift, "Is the picture shifted horizontally (to
         the right or left)?").
template(vertical_shift, "Is the picture shifted vertically (upwards
         or downwards)?").
template(colour_seen_on_screen, "Do you see any colour at all on
         the screen?").
template(type_of_colour_problem, "Which of the following colour
         problems do you think your TV set has?").
template(unsteadiness, "Which of the following types of unsteady
         pictures do you see on the screen?").
template(rolling_direction, "What is the direction in which the
         picture rolls over the screen?").
template(type_of_jumping, "How does the picture jump?").
template(sound_distorts_with_picture, "Is the distortion in sound
         accompanied by disturbances in the picture?").
template(plug_point_supply, "Is the plug point for your TV set, and
         the connecting cord, in good condition and supplying
         power?").
```

Menu Definitions

```
menu(picture_seen_on_screen, [yes, no]).
menu(type_of_picture_problem, [colour, dimension,
                               brightness, contrast, steadiness]).
menu(dark_screen, [yes, no]).
menu(complete_silence, [yes, no]).
menu(panel_lights, [on, off]).
menu(colour_seen_on_screen, [yes, no]).

menu(type_of_colour_problem, [pale_colours, incorrect_colours,
                               intermittent_colours, colour_patch]).
menu(type_of_raster, [black&white_lines_or_bars,
                      plain_white_screen,none]).
menu(rolling_direction, [vertical, horizontal]).
menu(unsteadiness, [rolling_picture, jumping_picture]).
```

```
menu(type_of_jumping, [continuously, irregularly]).
```

Note Definitions

```
note(id13, " Your TV is not getting power from the mains.\n You
     should check the following in order:\n the TV power cord, the
     power socket, the mains fuse.\n" ).
note(id14, " The internal power supply of the TV could be dead.\n But
     you should check the TV fuse first.\n" ).
note(id15, " Your TV is dead, but is getting power from the mains.\n
     The fault is most probably in the horizontal section.\n But
     you should check the TV's internal power supply first.\n" ).
note(id41, " Lines, dots and nettings are caused by interference.\n
     This is caused by pick-up from the antenna or the mains." ).
note(id52, " You can try to reduce the interference by changing the\n
     direction/location of the antenna.\n" ).
```

The Rules

Rules for faulty_module

```
rule(id19, faulty_module, antenna_or_tuner, 0.8) :-
     is(problem_with_picture, true),
     is(picture_seen_on_screen, no),
     is(snowy_picture, true),
     is(sound_absent, true).

rule(id30, faulty_module, antenna_or_tuner, 0.7) :-
     is(suspect_ante_tuner, true),
     is(snowy_picture, true).

rule(id35, faulty_module, antenna_or_tuner, 0.7) :-
     is(contrast_problem, true),
     is(snowy_picture, true).

rule(id42, faulty_module, antenna_or_tuner, 0.8) :-
     is(problem_with_picture, true),
     is(picture_seen_on_screen, yes),
     is(Ghosts_seen, true).

rule(id50, faulty_module, antenna_or_tuner, 0.8) :-
     is(jumping_problem, true),
     is(type_of_jumping, irregularly).



rule(id13, faulty_module, mains_supply, 0.4) :-
     is(TV_dead, true),
     is(panel_lights, off),
     isnot(plug_point_supply, true).

rule(id14, faulty_module, internal_power_supply, 0.4) :-
     is(TV_dead, true),
```

```
     is(panel_lights, off),
     is(plug_point_supply, true).

rule(id15, faulty_module, hor_sect, 0.4) :-
     is(TV_dead, true),
     is(panel_lights, on).

rule(id18, faulty_module, hor_sect, 0.8) :-
     is(problem_with_picture, true),
     is(raster_present, true),
     isnot(snowy_picture, true),
     is(type_of_raster, black&white_lines_or_bars).

rule(id21_1, faulty_module, hor_sect, 0.8) :-
     is(raster_absent, true),
     is(sound_present, true).

rule(id24, faulty_module, hor_sect, 0.7) :-
     is(size_problem, true),
     is(width_problem, true).

rule(id43, faulty_module, hor_sect, 0.7) :-
     is(shift_problem,true),
     is(horizontal_shift, true).

rule(id39, faulty_module, hor_sect, 0.25) :-
     is(brightness_problem, true).

rule(id39_1, faulty_module, hor_sect, 0.5) :-
     is(brightness_problem, true),
     isnot(snowy_picture, true).

rule(id46, faulty_module, hor_sect, 0.8) :-
     is(rolling_problem, true),
     is(rolling_direction, horizontal).

rule(id23, faulty_module, vert_sect, 0.7) :-
     is(size_problem, true),
     is(height_problem, true).

rule(id44, faulty_module, vert_sect, 0.7) :-
     is(shift_problem, true),
     is(vertical_shift, true).



rule(id47, faulty_module, vert_sect, 0.8) :-
     is(rolling_problem, true),
     is(rolling_direction, vertical).

rule(id49, faulty_module, vert_sect, 0.7) :-
     is(jumping_problem, true),
     is(type_of_jumping, continuously).

rule(id17, faulty_module, sync_ckt, 0.8) :-
     is(suspect_sync_ckt, true),
```

```
     is(raster_present, true),
     isnot(snowy_picture, true),
     is(type_of_raster, black&white_lines_or_bars),
     is(sound_absent, true).

rule(id20, faulty_module, IF_amp, 0.8) :-
     is(suspect_IF_amp, true),
     is(raster_present, true),
     isnot(snowy_picture, true),
     is(type_of_raster, plain_white_screen).

rule(id20_1, faulty_module, IF_amp, 0.5) :-
     is(suspect_IF_amp, true),
     is(raster_present, true),
     is(sound_present, true).

rule(id21_2, faulty_module, IF_amp, 0.4) :-
     is(suspect_IF_amp, true),
     is(raster_absent, true),
     is(sound_present, true).

rule(id28, faulty_module, IF_amp, 0.4) :-
     is(suspect_IF_amp, true),
     isnot(snowy_picture, true).

rule(id29, faulty_module, colour_proc_ckt, 0.8) :-
     is(suspect_colour_ckt, true),
     isnot(snowy_picture, true).

rule(id34, faulty_module, magnetisation, 0.9) :-
     is(colour_problem, true),
     is(colour_seen_on_screen, yes),
     is(type_of_colour_problem, colour_patch).

rule(id36_1, faulty_module, luma_proc_ckt, 0.4) :-
     is(suspect_luma_ckt, true),
     isnot(snowy_picture, true).

rule(id37, faulty_module, luma_proc_ckt, 0.6) :-
     is(suspect_luma_ckt, true),
     isnot(snowy_picture, true),
     is(contrast_too_high, true).


rule(id40_0, faulty_module, picture_tube, 0.5) :-
     is(suspect_picture_tube, true),
     isnot(snowy_picture, true),
     is(brightness_too_low, true).

rule(id53_1, faulty_module, picture_tube, 0.25) :-
     is(suspect_picture_tube, true),
     isnot(snowy_picture, true),
     ge(TV_age, 8.0).

rule(id40_1, faulty_module, picture_tube, 0.7) :-
     is(suspect_picture_tube, true),
```

```
     isnot(snowy_picture, true),
     is(brightness_too_low, true),
     ge(TV_age, 8.0).

rule(id41, faulty_module, interference, 0.8) :-
     is(problem_with_picture, true),
     is(picture_seen_on_screen, yes),
     is(Lines_Dots_Netting_seen, true).

rule(id52, faulty_module, interference, 0.8) :-
     is(problem_with_sound, true),
     is(sound_present, true),
     is(raster_present, true),
     is(sound_distorts_with_picture, true).

rule(id51_2, faulty_module, sound_proc_ckt, 0.8) :-
     is(suspect_sound_ckt, true),
     is(sound_present, true),
     is(raster_present, true),
     isnot(sound_distorts_with_picture, true).

rule(id51_3, faulty_module, sound_proc_ckt, 0.6) :-
     is(suspect_sound_ckt, true),
     is(sound_absent, true),
     is(raster_present, true).

rule(id51_4, faulty_module, speaker, 0.5) :-
     is(problem_with_sound, true),
     is(sound_absent, true),
     is(raster_present, true).
```

Rules for raster_present and sound_present

```
rule(id01_1, raster_present, true, 1.0) :-
     is(problem_with_picture, true),
     is(picture_seen_on_screen, yes).

rule(id01_3, raster_present, true, 1.0) :-
     is(problem_with_picture, true),
     is(picture_seen_on_screen, no),
     is(dark_screen, no).

rule(id01_4, raster_present, true, 1.0) :-
     isnot(problem_with_picture, true).

rule(id01_5, raster_absent, true, 1.0) :-
     is(problem_with_picture, true),
     is(picture_seen_on_screen, no),
     is(dark_screen, yes).

rule(id02_1, sound_absent, true, 1.0) :-
     is(problem_with_sound, true),
     is(complete_silence, yes).

rule(id02_2, sound_present, true, 1.0) :-
```

```
     is(problem_with_sound, true),
     is(complete_silence, no).

rule(id02_3, sound_present, true, 1.0) :-
     isnot(problem_with_sound, true).
```

Rules for snowy_picture

```
rule(id03_1, snowy_picture, true, 1.0) :-
     is(problem_with_picture, true),
     is(raster_present, true),
     is(snow_on_screen, true).

rule(id03_2, snowy_picture, true, -1.0) :-
     is(problem_with_picture, true),
     is(raster_present, true),
     isnot(snow_on_screen, true).

rule(id03_3, snowy_picture, true, -1.0) :-
     isnot(problem_with_picture, true).
```

Rules for TV_dead

```
rule(id12, TV_dead, true, 1.0) :-
     is(raster_absent, true),
     is(sound_absent, true).

rule(id12_1, TV_dead, true, -1.0) :-
     is(raster_present, true).

rule(id12_2, TV_dead, true, -1.0) :-
     is(sound_present, true).
```

Rules for suspect_luma_ckt

```
rule(lpc1, suspect_luma_ckt, true, 0.5) :-
     is(contrast_problem, true).

rule(lpc2, suspect_luma_ckt, true, 0.5) :-
     is(brightness_problem, true).
```

Rules for suspect_picture_tube

```
rule(id40_2, suspect_picture_tube, true, 0.7) :-
     is(brightness_problem, true).
```

Rules for suspect_sound_ckt

```
rule(spc1, suspect_sound_ckt, true, 0.7) :-
     is(problem_with_sound, true).
```

Rules for suspect_colour_ckt

```
rule(cpc1, suspect_colour_ckt, true, 0.7) :-
     is(colour_problem, true),
     is(colour_seen_on_screen, no).

rule(cpc2, suspect_colour_ckt, true, 0.7) :-
     is(colour_problem, true),
     is(colour_seen_on_screen, yes),
     is(type_of_colour_problem, pale_colours).

rule(cpc3, suspect_colour_ckt, true, 0.7) :-
     is(colour_problem, true),
     is(colour_seen_on_screen, yes),
     is(type_of_colour_problem, incorrect_colours).

rule(cpc4, suspect_colour_ckt, true, 0.7) :-
     is(colour_problem, true),
     is(colour_seen_on_screen, yes),
     is(type_of_colour_problem, intermittent_colours).
```

Rules for suspect_IF_amp

```
rule(ifa1, suspect_IF_amp, true, 0.4) :-
     is(contrast_problem, true).

rule(ifa2, suspect_IF_amp, true, 0.7) :-
     is(colour_problem, true),
     is(colour_seen_on_screen, no).

rule(ifa3, suspect_IF_amp, true, 0.7) :-
     is(colour_problem, true),
     is(colour_seen_on_screen, yes),
     is(type_of_colour_problem, pale_colours).

rule(ifa4, suspect_IF_amp, true, 0.7) :-
     is(problem_with_picture, true),
     is(picture_seen_on_screen, no).
```

Rules for suspect_sync_ckt

```
rule(sc1, suspect_sync_ckt, true, 0.6) :-
     is(problem_with_picture, true),
     is(picture_seen_on_screen, no).
```

Rules for suspect_ante_tuner

```
rule(at1, suspect_ante_tuner, true, 0.5) :-
     is(colour_problem, true),
     is(colour_seen_on_screen, no).

rule(at2, suspect_ante_tuner, true, 0.5) :-
     is(colour_problem, true),
     is(colour_seen_on_screen, yes),
```

```
     is(type_of_colour_problem, pale_colours).

rule(at3, suspect_ante_tuner, true, 0.5) :-
     is(colour_problem, true),
     is(colour_seen_on_screen, yes),
     is(type_of_colour_problem, intermittent_colours).
```

Rules for Picture Problems

```
rule(id22b, colour_problem, true, 1.0) :-
     is(problem_with_picture, true),
     is(picture_seen_on_screen, yes),
     is(type_of_picture_problem, colour).

rule(id22a, dimension_problem, true, 1.0) :-
     is(problem_with_picture, true),
     is(picture_seen_on_screen, yes),
     is(type_of_picture_problem, dimension).

rule(id22c, brightness_problem, true, 1.0) :-
     is(problem_with_picture, true),
     is(picture_seen_on_screen, yes),
     is(type_of_picture_problem, brightness).

rule(id22d, contrast_problem, true, 1.0) :-
     is(problem_with_picture, true),
     is(picture_seen_on_screen, yes),
     is(type_of_picture_problem, contrast).

rule(id45, rolling_problem, true, 1.0) :-
     is(problem_with_picture, true),
     is(picture_seen_on_screen, yes),
     is(type_of_picture_problem, steadiness),
     is(unsteadiness, rolling_picture).

rule(id48, jumping_problem, true, 1.0) :-
     is(problem_with_picture, true),
     is(picture_seen_on_screen, yes),
     is(type_of_picture_problem, steadiness),
     is(unsteadiness, jumping_picture).
```

Rule for size_problem

```
rule(id22f, size_problem, true, 1.0) :-
     is(dimension_problem, true),
     is(picture_size_abnormal, true).
```

Rule for shift_problem

```
rule(id22e, shift_problem, true, 1.0) :-
     is(dimension_problem, true),
```

```
is(picture_misaligned,true).
```

C.2 A Sample Run

The following is the log of a consultation session with GoodShow. The user responses are shown in **bold**.

Enter goal attribute (quit: main menu; ¡CR¿: default=faulty_module):
Enter the response filename(¡CR¿ if none):

Does the TV have any problem with the picture?
>> **0.60**

Do you see any picture at all on the screen?
>> **yes**

Which of the following display characteristics show a problem?
>> **colour**
CF= **0.60**
>> **steadiness**
CF = **0.60**

Do you see any colour at all on the screen?
>> **yes**

Which of the following colour problems do you think your TV set has?
>> **pale_colours**
CF= **0.60**

Is the screen full of tiny coloured spots?
>> **-0.60**

Do you see ghost images (multiple shadows of the pictures) on the screen?
>> **-0.60**

Which of the following types of unsteady pictures do you see on the screen?
>> **jumping_picture**

How does the picture jump?
>> **irregularly**

Does the TV have any problem with the sound?
>>**-0.60**

Which of the following types of raster scans do you see on the screen?
>> **none_of_the_above**

Do you see stray lines, dots or netting patterns on the screen?
>> **-0.60**

**** Conclusions ****

*The following modules are probably faulty in your TV set:

antenna_or_tuner	CF= 0.48
colour_proc_ckt	CF= 0.34
IF_amp	CF= 0.17

D. ConTax: Rule Base and Sample Run

D.1 Rule Base for Computing Taxable Salary

Type Definitions

```
type(total_income,f).
type(basic,f).
type(da,f).
type(hra,f).
type(cca,f).
type(annual_basic,f).
type(annual_da,f).
type(annual_hra,f).
type(annual_cca,f).
type(shift_al,f).
type(nshifts,i).
type(bonus,f).
type(other_inc,f).
type(sex,p).
type(income_in_hand,f).
type(standard_deduction,f).
type(std_ded_ceiling,f).
type(std_ded_calculated,f).
type(prof_tax,f).
type(monthly_prof_tax,f).
type(taxable_sal,f).
type(accom,p).
```

Goal Definition

```
goal(taxable_sal).
```

Template Definitions

```
template(sex,"What is the sex of the assessee? ").
template(basic,"What is your monthly Basic pay in Rs? ").
template(da,"What is your monthly Dearness Allowance in Rs? ").
template(hra,"What is your monthly House Rent Allowance in Rs? ").
template(cca,"What is your City Compensatory Allowance in Rs? ").
```

```
template(nshifts,"What is the estimated number of shifts you put in
         per year (zero if not applicable)? ").
template(bonus,"What is the amount of bonus received for the year
         in Rs? ").
template(other_inc,"What is the amount of other income for the year
         in Rs? ").
template(monthly_prof_tax,"What is the amount of Professional Tax
         deducted per month in Rs? ").
template(accom, "Do you stay in a residence allotted by NCST? ").
```

Menu Definitions

```
menu(sex,[female,male]).
menu(accom,[NCST,any-other]).
```

Compute Definitions

```
compute(taxsal1,taxable_sal, (income_in_hand - standard_deduction)).

compute(inc2,income_in_hand,(total_income - prof_tax)).

%% Total Annual Income
compute(inc1,total_income, (annual_basic+annual_da+annual_hra+
              annual_cca+shift_al+bonus+other_inc)).

%% Annual Income
compute(annual1,annual_basic,(12 * basic)).
compute(annual2,annual_da,(12 * da)).
compute(annual3,annual_hra,(12 * hra)).
compute(annual4,annual_cca,(12 * cca)).
compute(annual5,prof_tax,(12 * monthly_prof_tax)).

%% DA rules
compute(da1,da,0.83*basic) :-
        lt(basic,3500).
compute(da2,da,max(0.62*basic,2905)) :-
        ge(basic,3500), lt(basic,6000).
compute(da3,da,max(0.54*basic,3720)) :-
        ge(basic,6000).

%%  HRA rules
compute(hra1, hra, 0) :- is(accom,NCST).
compute(hra2, hra, min(0.3*(basic+da),1600)).

%% Shift Allowance
compute(sa1,shift_al,(basic+da)*nshifts/30).

compute(ms1, standard_deduction,
        min(std_ded_ceiling,std_ded_calculated)).

compute(std1,std_ded_ceiling,12000) :-
        is(sex,male).
compute(std2,std_ded_ceiling,15000):-
        is(sex,female),
```

```
        le(total_income,75000).
compute(std3,std_ded_ceiling,12000):-
        is(sex,female),
        gt(total_income,75000).

compute(std4,std_ded_calculated, ((33 * total_income)/100)).
```

D.2 Rule Base for Computing Tax

Type Definitions

```
type(basic,f).  % comes from the taxable salary module
type(da,f).     % comes from the taxable salary module
type(basic_da,f).
type(rebate,f).
type(tax1,f).
type(tax,f).
type(surcharge,f).
type(final_tax,f).
type(total_deductions,f).
type(taxable_sal,f).  % comes from the taxable salary module
type(investments,m).
type(sec88_deductions,f).
type(PF,f).
type(ELSS,f).
type(VOL_PF,f).
type(LIC,f).
type(PPF,f).
type(CTD,f).
type(NSC_viii,f).
type(Int_NSC_vii,f).
type(Group_ins,f).
type(NSS,f).
type(ULIP,f).
type(Jeevan_Dhara,f).
type(Jeevan_Akshay,f).
type(monthly_PF,f).
type(monthly_Group_ins,f).
type(inv_Int_NSC_vii,f).
type(monthly_VOL_PF,f).
type(inv_LIC,f).
type(inv_PPF,f).
type(inv_ELSS,f).
type(inv_CTD,f).
type(inv_NSC_viii,f).
type(inv_NSS,f).
type(inv_ULIP,f).
type(inv_Jeevan_Dhara,f).
type(inv_Jeevan_Akshay,f).
type(house_loan,f).
```

Goal Definition

```
goal(final_tax).
```

Template Definitions

```
template(investments,"What are the investments made
         in the current year?").
template(monthly_PF,"What is the amount of monthly Provident Fund
         deducted in Rs\n (excluding Voluntary PF)?").
template(monthly_VOL_PF,"What is the amount of monthly contribution
         to Voluntary Provident Fund in Rs?").
template(LIC,"What is the amount of LIC premium paid during
         the year in Rs?").
template(inv_PPF,"What is the amount invested in Public
         Provident Fund?").
template(inv_CTD,"What is the amount invested in Cumulative
         Time Deposits?").
template(inv_NSC_viii,"What is the amount invested in NSC_viii?").
template(Int_NSC_vii,"Is there any interest on NSC-vii included
         in your other income?\n If so enter the amount (0 otherwise):").
template(monthly_Group_ins,"Are you participating in Group insurance?
        If so \n enter the monthly amount (0 otherwise):").
template(inv_NSS,"What is the amount invested in NSS?").
template(inv_ULIP,"What is the amount invested in ULIP?").
template(inv_Jeevan_Dhara,"What is the amount invested in
         Jeevan_Dhara?").
template(inv_Jeevan_Akshay,"What is the amount invested in
         Jeevan_Akshay?").
template(inv_ELSS,"What is the amount invested in ELSS
         (Equity Linked \n Saving Schemes)?").
template(house_loan,"What is the amount (in Rs.) of house
         loan repayment made during the year?").
```

Menu Definitions

```
menu(investments,[PPF,CTD,NSC_viii,ULIP,NSS,
                 Jeevan_Dhara,Jeevan_Akshay,ELSS]).
```

Compute Definitions

```
compute(nettax1,final_tax,(tax + surcharge)) :-
        gt(tax,0).
compute(nettax2,final_tax,0).

compute(tax5,tax,(tax1 - rebate)).

compute(tax1,tax1,0):-
        le(taxable_sal,28000).
compute(tax2,tax1,(0.2 * (taxable_sal - 28000))):-
        gt(taxable_sal,28000),
        le(taxable_sal,50000).
compute(tax3,tax1, (4400 + (0.3 * (taxable_sal - 50000)))):-
```

```
        gt(taxable_sal,50000),
        le(taxable_sal,100000).
compute(tax4,tax1, (19400 + (0.4 * (taxable_sal - 100000)))):-
        gt(taxable_sal,100000).

%% Rebate
compute(reb1,rebate,(20 * total_deductions /100)) :-
        lt(total_deductions,60000).
compute(reb2,rebate,12000):-
        ge(total_deductions,60000).

%% Deductions
compute(ded2,total_deductions, (sec88_deductions + 10000)) :-
        ge(house_loan,10000).
compute(ded3,total_deductions, (sec88_deductions + house_loan)):-
        lt(house_loan,10000).
compute(ded1,sec88_deductions, (PF + VOL_PF + LIC + Int_NSC_vii +
        Group_ins + PPF + CTD + NSC_viii + ELSS +
        ULIP + NSS + Jeevan_Dhara + Jeevan_Akshay)).

compute(inv1,PF,(12 * monthly_PF)).
compute(inv2,Group_ins,(12 * monthly_Group_ins)).
compute(inv2a,VOL_PF,(12 * monthly_VOL_PF)).

compute(inv3,PPF,inv_PPF) :-
        is(investments,PPF).
compute(inv4,PPF,0).
compute(inv5,CTD,inv_CTD) :-
        is(investments,CTD).
compute(inv6,CTD,0).
compute(inv7,NSC_viii,inv_NSC_viii) :-
        is(investments,NSC_viii).
compute(inv8,NSC_viii,0).
compute(inv9,ULIP,inv_ULIP) :-
        is(investments,ULIP).
compute(inv10,ULIP,0).
compute(inv11,NSS,inv_NSS) :-
        is(investments,NSS).
compute(inv12,NSS,0).
compute(inv13,Jeevan_Dhara,inv_Jeevan_Dhara) :-
        is(investments,Jeevan_Dhara).
compute(inv14,Jeevan_Dhara,0).
compute(inv15,Jeevan_Akshay,inv_Jeevan_Akshay)  :-
        is(investments,Jeevan_Akshay).
compute(inv16,Jeevan_Akshay,0).
compute(inv17,ELSS,inv_ELSS) :-
        is(investments,ELSS).
compute(inv18,ELSS,0).

compute(g1,monthly_Group_ins,45.0) :-
        lt(basic_da,1500).
compute(g2,monthly_Group_ins,75.0) :-
        ge(basic_da,1500),
        lt(basic_da,2000).
compute(g3,monthly_Group_ins,100.0) :-
        ge(basic_da,2000).
```

```
compute(bd,basic_da,basic+da).

%% Surcharge
compute(surch1,surcharge,(tax * 12 /100)) :-
        gt(taxable_sal,100000),
        gt(tax,0).
compute(surch2,surcharge,0).
```

D.3 Rule Base for Investment Rating

Type Definitions

```
type(rating,b).
type(interest_exemption,p).
type(wealth_tax_exemption,p).
type(investment,p).
type(investment_risk,f).
type(user_risk_preference,i).
type(investment_pledge_ability,p).
type(investment_ceiling,f).
type(user_loan_requirement,p).
type(risk,i).
type(liquidity,i).
type(investment_name,p).
type(investment_rate,f).
type(rebate_u/s_88,p).
type(investment_lockup,f).
type(user_insurance_requirement,p).
type(user_insurance_amount,f).
type(approaching_retirement,p).
type(user_liquidity_preference,f).
type(pay_back_preference,p).
type(age,i).
```

Goal Definition

```
goal(rating).
```

Template Definitions

```
template(age,"What is your age in years?").
template(user_risk_preference,"Enter the risk preference with a scale
         from 1-10 (where 1 is low and 10 is high):").
template(user_loan_requirement,"Do you intend to take a loan pledging
         your securities in the near future?").
template(user_insurance_requirement,"Do you need to insure yourself or
         any member of your family\n i.e., do you intend to take a new
         policy this year?").
template(user_insurance_amount,"What s the amount for which you intend
```

```
        to insure (in Rs.)?").
template(approaching_retirement,"Is your taxable income going to drop
        substantially in the near future \n(for e.g. if you are
        going to retire)?").
template(user_liquidity_preference,"Enter the number of years within
        which you expect to withdraw your investment.\n Different
        schemes lockup your investment for different periods:").
template(pay_back_preference,"Do you prefer the return from the
        investment as a lump sum pay back,\n or would you like a
        monthly payment somewhat like a pension?").
```

Initial Definition

```
initial([investment_name]).
```

Menu Definitions

```
menu(interest_exemption,[totally,u/s_80L,u/s_88,no]).
menu(approaching_retirement,[yes,no]).
menu(user_risk_preference,[1,10]).
menu(wealth_tax_exemption,[totally,upto_5_lakhs,no]).
menu(user_loan_requirement,[yes,no]).
menu(user_insurance_requirement,[yes,no]).
menu(pay_back_preference,[lump_sum,monthly]).
```

Text Database Interface Definitions

```
textdb(investment_rate, "investments.db",investment_name,2).
textdb(investment_lockup, "investments.db",investment_name,3).
textdb(rebate_u/s_88, "investments.db",investment_name,4).
textdb(interest_exemption, "investments.db",investment_name,5).
textdb(wealth_tax_exemption, "investments.db",investment_name,6).
textdb(investment_risk, "investments.db",investment_name,7).
textdb(investment_pledge_ability, "investments.db",investment_name,8).
textdb(investment_ceiling, "investments.db",investment_name,9).
```

Compute Definitions

```
compute(c1, risk, (investment_risk-user_risk_preference)).
compute(c2,liquidity, (user_liquidity_preference-investment_lockup)).
```

The Rules

```
%% General rules applicable for all investments

rule(interest1,rating,true,0.2) :-
     is(interest_exemption,u/s_80L).
rule(interest2,rating,true,0.25) :-
     is(interest_exemption,totally).
rule(interest3,rating,true,0.225) :-
     is(interest_exemption,u/s_88).
rule(interest4,rating,true,-0.1) :-
```

```
     is(interest_exemption,no).

rule(wealth1,rating,true,0.2) :-
     is(wealth_tax_exemption,upto_5_lakhs).
rule(weatlh2,rating,true,0.25) :-
     is(wealth_tax_exemption,totally).
rule(weatlh3,rating,true,-0.1) :-
     is(wealth_tax_exemption,no).

rule(pledge_ability1,rating,true,0.2) :-
     ge(user_risk_preference,0),
     ge(investment_ceiling,0),
     is(user_loan_requirement,yes),
     is(investment_pledge_ability,yes).
rule(pledge_ability2,rating,true,-0.3) :-
     is(user_loan_requirement,yes),
     is(investment_pledge_ability,no).

rule(risk1,rating,true,0.2) :-
     ge(investment_risk,0),
     ge(user_risk_preference,0),
     lt(risk,0).
rule(risk2,rating,true,0.3) :-
     eq(risk,0).
rule(risk3,rating,true,-0.2) :-
     gt(risk,0).

rule(lockup1,rating,true,-0.3) :-
     ge(investment_lockup,0),
     ge(user_liquidity_preference,0),
     lt(liquidity,0).
rule(lockup2,rating,true,0.1) :-
     ge(investment_lockup,0),
     ge(user_liquidity_preference,0),
     eq(liquidity,0).
rule(lockup3,rating,true,0.1) :-
     ge(investment_lockup,0),
     ge(user_liquidity_preference,0),
     gt(liquidity,0).




%%  Investment Specific Rules

% Withdrawals are not taxable & can be
% redeposited for exemption u/s 88.
rule(ISR_PPF,rating,true,0.25) :-
     is(investment_name,PPF).


% More preference given to ULIP if the insurance
% needs of the employee is less than Rs.1,20,000.
rule(ISR_INSURANCE_1,rating,true,0.15) :-
     is(investment_name,ULIP),
     is(user_insurance_requirement,yes),
```

```
     le(user_insurance_amount,120000).

% Rating is reduced if the insurance needs are
% more than Rs.1,20,000 since ULIP has limit
% upto 60,000
rule(ISR_INSURANCE_3,rating,true,0.10) :-
     is(investment_name,ULIP),
     is(user_insurance_requirement,yes),
     gt(user_insurance_amount,120000).

% Investment in Jeevan_Akshay is is preferable
% if the taxable income when the annuity begins
% is expected to be low. (i.e. after retirement)
rule(ISR_JEEVAN_AK1,rating,true,0.20) :-
     is(investment_name,Jeevan_Akshay),
     is(approaching_retirement,yes).

% Jeevan Akshay is applicable for those above 50
% years in age only
rule(ISR_JEEVAN_AK2,rating,true,-1.0) :-
     is(investment_name,Jeevan_Akshay),
     lt(age,50).

% The next four rules are for the pension payment
% preferences of the user.
rule(MONTHLY_VS_LUMP1,rating,true,0.1) :-
     is(investment_name,Jeevan_Dhara),
     is(pay_back_preference,monthly).

rule(MONTHLY_VS_LUMP2,rating,true,0.1) :-
     is(investment_name,Jeevan_Akshay),
     is(pay_back_preference,monthly).

rule(MONTHLY_VS_LUMP3,rating,true,-0.1) :-
     is(investment_name,Jeevan_Dhara),
     is(pay_back_preference,lump_sum).

rule(MONTHLY_VS_LUMP4,rating,true,-0.1) :-
     is(investment_name,Jeevan_Akshay),
     is(pay_back_preference,lump_sum).
```

D.4 A Sample Run

This chapter gives the sample run of ConTax. The text in *this font are annotations on the log.* Normal text font like this indicates messages/prompts by ConTax and **text in this font is user response**.

Do you need information on how to use ConTax
if yes press y or Y, otherwise press any other key: **n**

If the user types in yes, an initial help on what ConTax *is and how to use it is provided.*

Now the taxable salary module *is invoked to evaluate the taxable salary.*

What is the sex of the assessee? (MENU)
: male

What is your monthly Basic pay in Rs?
4000

What is your monthly Dearness Allowance in Rs?
500

What is your monthly House Rent Allowance in Rs?
800

What is your City Compensatory Allowance in Rs?
250

What is your shift allowance for the year in Rs?
0

What is the amount of bonus received for the year in Rs?
0

What is the amount of other income for the year in Rs?
0

What is the amount of Professional Tax deducted per month in Rs?
50

The questions above are put to the user by the taxable salary module*. If there is an interface with the database containing the salary details, this information can be directly retrieved from the database rather than asking the user interactively.*

After this the investment rating module *takes over. If a response file which has the input information for this module already exists in the directory from which* ConTax *is invoked, then the following question is asked to the user:*

Do you want to use the existing details in the cust.res file for investment preferences. Press y or Y for yes, press any other key for no:
n

Enter the risk preference with a scale from 1 to 10 (where 1 is low and 10 is high)
:
6

Do you intend to take a loan pledging your securities in the near future? (MENU)
yes

Enter the number of years within which you expect to withdraw your investment. Different schemes lockup your investment for different periods.
3

Do you prefer the return from the investment as a lump sum pay back or would you like a monthly payment somewhat like a pension? (MENU)
monthly

Is your taxable income going to drop substantially in the near future (for eg, if you are going to retire)? (MENU)
yes

What is your age in year?
30

Do you need to insure yourself or any member of your family i.e., Do you intend to take a new policy this year? (MENU)
yes

What is the amount for which you intend to insure (in Rs)?
30000

Getting the above inputs the Invexp *module calculates the rating. After this the* tax liability module *takes over. The following inputs are for* tax liability module.

What is the amount (in Rs) of house loan repayment made during the year?
5000

What is the amount of monthly Provident Fund deducted in Rs (excluding Voluntary PF)?
500

What is the amount of monthly contribution to Voluntary PF in Rs?
50

What is the amount of LIC premium paid during the year in Rs?
500

Is there any interest on NSC-vii included in your other income? If so, enter the amount (0 otherwise):
0

Are you participating in Group insurance? If so, enter the monthly amount (0 otherwise):
0

What are the investments made in the current year? (MENU)
PPF
ULIP
end

What is the amount invested in Public Provident Fund?
5000

What is the amount invested in ULIP?
3000

After taking these inputs ConTax *displays the screen with spread sheet type interface. All the investments with their appropriate ratings and amounts are displayed. In the bottom of the screen the income of the employee, present tax liability, tax liability before the most recent change and the balance income are displayed.*

INVESTMENT NAME	RATING	CEILING	INVESTED	FINAL AMT
1. PPF	0.660000	60000	5000	5000
2. ULIP	0.560000	60000	3000	3000
3. NSC_viii	0.50000	10000	0	0
4. ELSS	0.490000	20000	0	0
5. NSC	0.400000	40000	0	0
6. CTD	0.330000	20000	0	0
7. Jeevan_Dhara	0.250000	60000	0	0
8. Jeevan_Akshay	-1.000000	60000	0	0
:			(Type ? for	help)

INCOME : 66000 TAX : 1580 (0)
TOTAL INVESTMENTS : 20100 BALANCE : 44320

To increase the amount of investment in PPF to Rs. 7500, the user has to type **1,+2500** *in the colon prompt. After this modification the screen would look like*

INVESTMENT NAME	RATING	CEILING	INVESTED	FINAL AMT
1. PPF	0.660000	60000	5000	7500
2. ULIP	0.560000	60000	3000	3000
3. NSC_viii	0.50000	10000	0	0
4. ELSS	0.490000	20000	0	0
5. NSC	0.400000	40000	0	0
6. CTD	0.330000	20000	0	0
7. Jeevan_Dhara	0.250000	60000	0	0
8. Jeevan_Akshay	-1.000000	60000	0	0
:			(Type ? for	help)

INCOME : 66000 TAX : 1080 (1580)
TOTAL INVESTMENTS : 22600 BALANCE : 42320

If the user is satisfied with the current position, that is the amount invested in various schemes and the current tax liability. The user can quit the screen by entering **q** *in the colon prompt. A final report is prepared. The final report prepared after the sample run is given on the next page.*

INCOME TAX CALCULATION FOR THE YEAR
APRIL.92 TO MARCH.93
==============================

(A)TOTAL INCOME.

1. Basic Salary	48000.00	
2. DA	6000.00	
3. HRA	9600.00	
4. CCA	3000.00	
5. Shift_Allowance	0.00	
6. Bonus	0.00	
7. Other_Income	0.00	
	Total Income Rs.	66600.00

(B) DEDUCTIONS UNDER INCOME TAX ACT

1. Standard deduction	12000.00	
2. Profession Tax (iii) u/s 16	600.00	
	Taxable Sal Rs.	54000.00

(C) DEDUCTIONS U/S 88 (6)(ii)(max. Rs.60,000)

Provident Fund	6000.00	
VOL. PF.	600.00	
LIC(Salary Savings Scheme)	500.00	
PPF	7500.00	
ULIP	3000.00	
Housing loan repayment	5000.00	
		22600.00

The rate of income tax applicable is 30% of taxable income over and above Rs.50,000 plus Rs.4,400.

The tax before rebate is	Rs. 5600.00
The rebate 20% of total of (C) is	Rs. 4520.00
The Income tax payable by the assessee is	Rs. 1080.00

E. Answers to Selected Exercises

Chapter 1

6. (a) Task of solving a set of simultaneous equations

   For solving a set of simultaneous equations, the expert systems approach is *not* suitable. The reasons include the following:

   - This task essentially involves arithmetic computations. The ES approach is aimed at tasks involving symbolic computations.
   - The task is completely procedural. The ES approach is designed to take advantage of the declarative knowledge of a problem area. Though the ES approach can be used to encode procedural knowledge, this task does not exploit the advantages of the ES approach, such as ease of understanding, modifiability, modularity, etc.
   - Thus, it is best that this task be tackled using conventional programming techniques, possible using a component of a library of mathematical software.

   (b) Task of computing the income tax

   The expert systems approach can be applied to the task of computing the income tax. The reasons include the following:

   - Computing income tax is inherently rule based and the knowledge used is mostly declarative in nature. The ES approach can exploit these features of the task.
   - The rules and regulations governing income tax computations are bound to change quite often. This would require regular modifications to the system. These modifications are easier to make in an expert system.

7. **A representation for the 8-puzzle problem**

   The list structure can be used as the representation for the 8-puzzle problem.

   The square is represented using a list. The elements of this list are the representations of the cells.

   A cell is represented as a list. This list consists of three elements. The first element denotes the position of the cell. For example, 1 for the first cell, 2 for the second cell, etc. The second element denotes the number of the tile contained by the cell. For example, 1 for the tile number one, 2 for the tile number two, etc. The third element is a list of cell positions which are *adjacent to* the current cell.

The relationship *adjacent to* is defined as follows: C1 is *adjacent to* C2, if a tile in C1 can be moved to C2, when C2 is empty.

Now refer to the square given in the figure with the exercise. This square can be represented using a list in the following way.

The first cell position is represented by the list:

(1 1 (2 4))

Similarly, the ninth cell position is represented by the list:

(9 nil (6 8))

The complete square can be represented by list of such lists as follows:

```
( (1 1 (2 4))
  (2 2 (1 3 5))
  (3 3 (2 6))
  (4 4 (1 5 7))
  (5 5 (2 4 6 8))
  (6 6 (3 5 9))
  (7 7 (4 8))
  (8 8 (5 7 9))
  (9 nil (6 8))
)
```

A solution strategy for the 8-puzzle problem

The above representation mechanism has been designed keeping in mind the possibility of using a search mechanism as the solution strategy. The list representing the adjacent cell positions are meant to facilitate traversal to the next state in the search space.

Because brute force search of the search space is likely to be an enormous and inefficient task, we need to employ a heuristic search mechanism as the solution strategy. One possible candidate is *best-first search*.

Best-first search requires a rank to be computable for each of the states of the search space associated with a possible next move. A number of options exist for designing the heuristic to determine the rank of a state. One simple option, in this problem, is to let the rank of a state be the number of tiles in correct positions in the puzzle.

Chapter 2

1. Rules are suitable for expressing knowledge which is characterised by situations and corresponding actions. It turns out that most of the

practical knowledge in real life can be viewed in this fashion. Therefore rules became quite a widely used formalism for knowledge representation in expert systems.

Apart from situation-action knowledge there are other kinds of knowledge as well. These include spatial knowledge and temporal knowledge. Rules are not suitable for representing these kinds of knowledge.

There is one other kind of knowledge which cannot be viewed in terms of situations and actions. This is the knowledge about a network of objects (or concepts) in which the objects are linked to one another. The types of links include *a-kind-of*, *is-a*, etc. This kind of knowledge can be represented using frame based systems which are described in Chapter 4.

2. The following is the list of possible ways the antecedents of rule *p1* will be instantiated.

   - hungry(gopal) **and** serves(balaji, dosa)
   - hungry(gopal) **and** serves(taj, puri)
   - hungry(gopal) **and** serves(balaji, idli)
   - hungry(sekar) **and** serves(balaji, dosa)
   - hungry(sekar) **and** serves(taj, puri)
   - hungry(sekar) **and** serves(balaji, idli)
   - hungry(sunita) **and** serves(balaji, dosa)
   - hungry(sunita) **and** serves(taj, puri)
   - hungry(sunita) **and** serves(balaji, idli)

   Using recency as the conflict resolution strategy will select the rule corresponding to the following instantiation to fire:

   hungry(sekar) **and** serves(taj, puri)

   This will result in the following element being added to the WM:

   goes-to(sekar, taj)

3. After using specificity as the conflict resolution strategy the conflict set will have the following instantiations:

   - hungry(sekar) **and** likes(sekar, puri) **and** serves(taj, puri)
   - hungry(sunita) **and** likes(sunita, idli) **and** serves(balaji, idli)
   - hungry(sunita) **and** likes(sunita, dosa) **and** serves(balaji, dosa)

   After using the textual order of the facts the following instantiation will qualify for firing:

   hungry(sekar) **and** likes(sekar, puri) **and** serves(taj, puri)

   This will result in the following element being added to the WM:

   goes-to(sekar, taj)

4. The final WM will contain the following elements:

   goes-to(sekar, taj)
   goes-to(sunita, balaji)
   goes-to(gopal, balaji)
   goes-to(gopal, taj)
   goes-to(sekar, balaji)
   goes-to(sunita, taj)

   According to the WM, Gopal will go to Balaji or Taj.

   If we use refractoriness first and specificity next, we end up in the same situation; Gopal will still go to Balaji or Taj.

7. (a) Designing the layout of a house

   Forward chaining is suitable for solving the problem of designing the layout of a house.

   Justification

   Forward chaining is suitable for applications where the solutions need to be constructed incrementally. The layout of a house is designed in an incremental fashion. Thus forward chaining can be applied to solve the problem of designing the layout of a house.

   (b) Diagnosing a patient for a disease

   Backward chaining is suitable for solving the problem of diagnosing a patient for a disease.

   Justification

   Backward chaining is suitable for problems whose solutions need to be incrementally refined. The task of diagnosing a patient for a disease requires the disease type to be incrementally refined until the treatable cause is identified. Thus the diagnosis task can be solved by the use backward chaining.

8. The following are two meta-rules which are applicable to the task of troubleshooting of TV receivers (See Chapter 12 for a case study of a TV troubleshooting system).

   Meta-rule 1

   If the audio is not normal and the video is normal then examine the rules for sound circuit faults.

   Meta-rule 2

   If there is no audio and video, then first examine the tuner section before examining the video signal processing section or the audio processing section.

Chapter 3

3. Let I = <W, f> be an interpretation. Let W = {1, 3, 5, 7, 9}. Let f(p) be the "less than" relation.

   For this interpretation, we can verify that for all possible term assignments, the two clauses are true.

That means that I is a model for the clauses.

4. The given clause is not a logical consequence of the earlier two clauses.

   Consider the interpretation given in *Exercise 3*. The given clause is not true for this interpretation under the following term assignment: f(X) = 3 and f(Y) = 5.

6. (a) X = a, Y = Z
   (b) The two expressions cannot be unified.
   (c) X = ram, Y = father(ram)

Chapter 4

3. The exact syntax used in this answer would depend on the frame representation system that is used. Here we have given a partial solution using an informal notation to emphasise the relevant concepts.

   The hierarchy of geometrical figures can be represented using frames in the following way.

   First we represent a Polygon, with N sides, and N angles. The convention used is that `side[1]` is the length of the first side (numbered in some contiguous order), `side[N]` is the length of the last side, `angle[1]` is the internal angle subtended by sides 1 and 2, `angle[N]` is the angle subtended by sides N and 1, etc. Each side is a real number, greater than 0.0. Each angle is a real number, within the range 0.0 to 180.0.

   Other frames are defined with calls to functions such as `get-value` or `sumsides`, which have to be defined in the frame language being used.

```
Frame Polygon is-a Thing
  number-of-sides  : :type integer       :greater 0
  number-of-angles : :type integer
                     :if-needed (get-value number-of-sides)
  side             : :type real-array
                     :greater 0.0
  angle            : :type real-array
                     :inrange  (0.0 180.0)
end Frame

Frame Quadrilateral is-a Polygon
  number-of-sides  : 4
  perimeter        : :if-needed '(sumsides 4)
  area             : ...
end Frame

Frame Parallelogram is-a Quadrilateral
  angle            : :restriction
                       (and
                          (equal angle[1] angle[3])
                          (equal angle[2] angle[4]))
```

```
  side             : :restriction
                        (and
                           (equal side[1] side[3])
                           (equal side[2] side[4]))
end Frame

Frame Rectangle is-a Parallelogram
  angle            : :restriction
                        (equal '(90.0 90.0 90.0 90.0))
  area             : :if-needed
                        (* (get-value side[1])
                           (get-value side[2]))
end Frame

Frame Square is-a Rectangle
  side             : :restriction (equal side[1] side[2])

end Frame
```

Chapter 5

1. Consider the various sources of uncertainty discussed in the chapter.

   Inaccuracy of instruments: Electronic measuring instruments are likely to be accurate enough. But faulty instruments which give wrong readings, cannot be ruled out.

   Imprecision of language: Except for end users reporting the symptoms of an equipment, an on-site diagnostician (who would be more of an expert) may not face this problem. However, if remote diagnosis is envisaged, the system has to rely on the user's responses for all data and hence this may become a problem. For example, a user may report that a chip is too hot when the temperature actually is well within tolerance limits.

   Incompleteness: Not all desired readings are available equally easily. Some of the readings may require opening up of the equipment or expensive tests and may not be possible. Hence diagnosis without full information is essential.

   Multiple sources of information: Where there are many observations available on the machine, consistency of the various readings should be considered.

4. Let P(D) : probability that a person has disease D
   and P(H) : probability that a person has a headache

   From the input:

P(D)	= 0.2
P(H)	= 0.45
P(D and H)	= 0.15

A person picked at random means that no specific information is available and hence, the answer will be the a priori probability P(D), i.e., 0.2.

When we consider only those having a headache, we are looking at the conditional probability P(D|H).

P(D|H) = P(D and H)/P(H)
 = 0.15 / 0.45
 = 0.33

Therefore, presence of headache increases the belief in expecting D.

5. Yes. The formula will be (P(d) - P(d & s)) / (1 - P(s)).

 Hint: If you consider two events A and B, P(A) can be interpreted considering B also. In the world of probability, any event either happens or does not. Hence for any given case, either B is true or not B is true.

 Hence the probability of A, P(A) can be viewed as the sum of:

 Probability of A and B happening + Probability of A and not B happening

 i.e., P(A) = P(A & B) + P(A & (not B)).

 Try using this relationship to derive the answer.

7. (a) i) No. only P1 and P3 will fire. The first condition in P2 has a CF¡0.2 (in this case it is negative).

 ii) No. Since P2 is ruled out based on the first condition, the second condition need not be examined. The conditions of P1 and P3 can be evaluated with the available information.

 iii) Contribution from the rule P1 = min(0.9) * 0.2 = 0.18
 Contribution from the rule P3 = min(0.4) * 0.25 = 0.1

 iv) Assume there are no other rules applicable in the rule base.
 CF of potential_threat, at the beginning = 0.0
 After P1: = 0.0 + 0.18 - 0.18 * 0.0
 = 0.18
 After P3: = 0.18 + 0.1 - 0.18 * 0.1
 = 0.262

 (b) Hint: Assume it is X and note that the order of rules is irrelevant as far as the final CF is concerned. Hence assume P1 fires last. Work out the resulting CF as done in (a). Solve the equation.

 Ans: The data is inconsistent. P2 and P3 alone give a contribution of 0.43 which is more than the total CF of potential_threat. Since P1's contribution will be positive, it cannot reduce the CF of potential_threat.

Chapter 6

2. *One rule is identified where a canned message is sufficient to provide an explanation to a* why *query from the user.*

Another rule is identified where a canned message is not sufficient as an explanation for a why *query from the user.*

Case 1, where a canned message is sufficient

Consider the following rule from *Hruday*.

```
rule(cf8, HD_confirmation, true, -0.40) :-
  is(dysp_on_exertion, true),
  is(lung_disease, true).
```

During the evaluation of this rule, the user might be inclined to ask *why*. While evaluating the second antecedent

```
  is(lung_disease, true)
```

the user might be slightly puzzled as to why the system is asking about a lung problem. Normally the user might expect only questions directly related to the heart. At this juncture the user might be prompted to ask *why*.

The following canned message can be displayed by the system in response to this *why* query:

```
I am trying to evaluate the following rule:

If dysp_on_exertion is true and
lung_disease is true then conclude that
HD_confirmation is true with CF = -0.4

For that, I need to evaluate the antecedent
"lung_disease is true". That is why I am
asking this question.
```

Case 2, where a canned message is not sufficient

Consider the following rule from Hruday to illustrate a case where a canned message may not be sufficient to answer *why* kind of questions.

```
rule(dis9, disease, CHD, 0.60) :-
  lt(age, 15.0),
  is(HD_confirmation, true),
  is(nail_blue, true).
```

Assume that the user is asking *why* when the system is evaluating the following antecedent:

```
  is(nail_blue, true)
```

The system can display the following canned message in response to this query:

```
I am trying to evaluate the following rule:

If age is less than 15 years and
HD_confirmation is true and
nail_blue is true then conclude that
disease is CHD with CF = 0.6

For that I need to evaluate the antecedent
"nail_blue is true". That is why I am asking
this question.
```

But this kind of explanation may not be the ideal thing the user wants, partly because this explanation refers to an attribute (*HD_confirmation*) which is internal to the rule base. The user is not concerned about this attribute, nor is he likely to understand its meaning or significance.

Instead, what is required is an explanation in terms of attributes which the user can easily associate with. In our specific case, instead of providing an explanation which refers to the attribute *HD_confirmation*, an explanation could be provided using attributes which contribute to the value of the parameter *HD_confirmation*. Then the explanation would be in terms of user understandable attributes, and therefore more meaningful.

Chapter 8

1. The following questions are aimed at eliciting domain knowledge from the expert.
   - What could be the cause of the screen going completely blank while the audio is normal?
   - What could be the reason for the screen getting filled with coloured dots and the audio getting distorted?
   - What are the symptoms which would be noticed when the antenna is not fixed properly?
   - What do you think is the most common problem encountered in a TV receiver?

Chapter 9

3. The same rule cannot be the cause of false positives and false negatives. Therefore, the same set of rules cannot lead to the 3 false positives and 3 false negatives. Therefore, S1 and S2 have to be disjoint.

Chapter 10

1. Proceed as in the example worked out in the chapter. You should get the tree shown in Figure E.1:

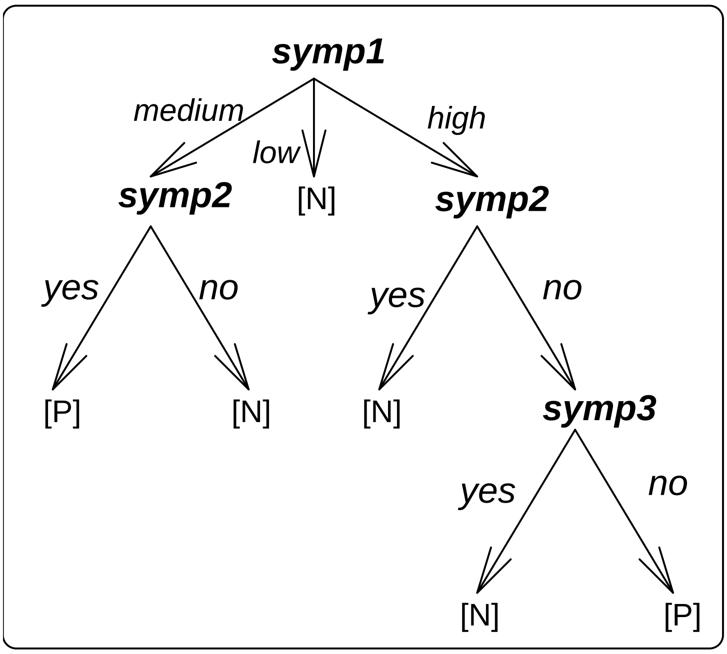

Figure E.1: Induction Tree for Exercise 1 of Chapter 10

2. Initial Distribution of beliefs is: {g,c,h,p} = 1.0; all else = 0.0.

   (a) M1 : {h,c} = 0.6; {g,c,h,p} = 0.4
   Combining with the initial distribution, we get {g,c,h,p} = 0.4; {h,c} = 0.6; all else = 0.0.

   (b) M2 : {c} = 0.7; {g,c,h,p} = 0.3
   Combined with the distribution from (a), we get {g,c,h,p} = 0.12; {c} = 0.7; {h,c} = 0.18; all else 0.0

   (c) M3 : {h,c,p} = 0.4; {h,c,g,p} = 0.6
   Combined with the distribution from (b), we get the final distribution:

   {c} = 0.7; {h,c,p} = 0.048; {h,c} = 0.18; {h,c,g,p} = 0.03; all else 0.0.

   The question requires the belief intervals of various subsets of F also. We will work out the belief interval for the set S = {h,c}. For this the complement set S' = {g,p}.

   At the start:

   Bel(S) = {h}+{c}+{h,c} = 0.0
   Bel(S') = {g}+{p}+{g,p} = 0.0

BI = [0,1] – no belief committed anywhere

After (a):

Bel(S) = 0.6; Bel(S') = 0.0
BI = [0.6, 1.0]

After (b):

Bel(S) = 0.7 + 0.18 = 0.88; Bel(S') = 0.0
BI = [0.88, 1.0]

After (c):

Bel(S) = 0.7 + 0.18 = 0.88; Bel(S') = 0.0
BI = [0.88, 1.0]

You can see that as the evidence for S increases the left border of the belief interval keeps moving to the right (towards 1.0). In this case, there was no evidence against S. Hence the right border stayed stationary at 1.0. The final uncertainty about S is 1.0 - 0.88 = 0.12.

3. A simple mechanism would be as follows. Note that we are not worrying about the efficiency of the algorithm. You may want to re-work the algorithm to improve the efficiency. This solution requires familiarity with the contents of Chapter 3 on Clausal Form Logic.

   The method:

   First we convert all these descriptions (the model description and the observations), into the general clausal form (i.e., c1,c2,c3 $\Leftarrow$ a1,a2,a3). This can be done by systematically simplifying the formula using distributive laws, till they become flat as in the clausal form.

   Alternately, you could assume this step is already done (i.e., all descriptions are provided in clausal form).

   Let the resulting set of clauses be called S.

   Now we apply the resolution procedure to check if the set as a whole is inconsistent. It must be. Otherwise, the device is behaving as per expectation from the model.

   Next comes the real Model Based Reasoning. We should try to find out every non-empty set of clauses, say A, from the set of clauses S such that:

   1. Every clause in A belongs to the model description component of S
   2. The set of clauses S excluding those in A, is consistent.
   3. If you delete any one clause from A, property 2 does not hold.

   This has to be done by systematic enumeration. *For simplicity, we assume each A will contain only one clause.* So we can proceed as follows:

```
PA = empty   {the set of As}
For i in model description clauses of S
```

```
begin
   S' = S - i;
   if S' is consistent add {i} to PA
end
return PA
```

Note that we are assuming here that the observations are all correct and the fault, if any, is with the equipment. Having found an A (i.e., members of PA) and assuming each clause was associated with some specific component of the equipment, you can retrieve the component(s) associated with A. A fault in this explains the symptoms observed.

References

[Aikins, 1983] Aikins JS. *Prototypical Knowledge for Expert Systems.* Artificial Intelligence, Vol 20, pp 163-210, 1983.

[Alpert and Rippe, 1980] Alpert JS and Rippe JM. *Manual of Cardiovascular Diagnosis and Therapy.* Asian Edition, Little, Brown and Company, Boston, 1980.

[Alty, 1987] Alty JL. *The Limitations of Rule Based Expert Systems.* In Jiri Kriz (ed), *Knowledge Based Expert Systems in Industry*, Ellis Horwood, pp 17-22, 1987.

[Anderson, 1983] Anderson JR. *The Architecture of Cognition.* Harvard University Press, 1983.

[Barr and Feigenbaum, 1982] Barr A and Feigenbaum EA (eds). *The Handbook of Artificial Intelligence, Vol II.* William Kaufman Inc, 1982.

[Bennett et al, 1978] Bennett JS, Creary L, Engelmore R and Melosh R. *SACON: A Knowledge Based Consultant for Structural Analysis.* Technical Report of the Heuristic Programming Project, HPP-78-23, Stanford University, 1978.

[Bobrow and Collins, 1975] Bobrow DG and Collins AM (eds). *Representation and Understanding: Studies in Cognitive Science.* Academic Press, 1975.

[Bobrow and Winograd, 1977] Bobrow DG and Winograd T. *An Overview of KRL, a Knowledge Representation Language.* Cognitive Science, Vol 1, No 1, pp 3-46, 1977. Also Chapter 13 in [Brachman and Levesque, 1985], pp 263-285.

[Brachman, 1979] Brachman RJ. *On the Epistemological Status of Semantic Networks.* In [Findler, 1979], pp 3-50. Also Chapter 10 in [Brachman and Levesque, 1985], pp 191-215.

[Brachman and Levesque, 1985] Brachman RJ and Levesque HJ. *Readings in Knowledge Representation.* Morgan Kaufmann, 1985.

[Brachman and Schmolze, 1985] Brachman RJ and Schmolze JG. *An Overview of the KL-ONE Knowledge Representation System.* Cognitive Science, Vol 9, No 2, pp 171-216, 1985.

[Brachman et al, 1983] Brachman RJ, Fikes RE and Levesque HJ. *KRYPTON: A Functional Approach to Knowledge Representation.* IEEE Computer, Vol 16, No 10, pp 67-73, 1983. A revised version of this appears as Chapter 24 in [Brachman and Levesque, 1985], pp 411-429.

[Bratko, 1986] Bratko I. *Prolog Programming for Artificial Intelligence.* Addison-Wesley, 1986.

[Brown et al, 1982] Brown JS, Burton RR and deKleer J. *Pedagogical, Natural Language and Knowledge Engineering Techniques in SOPHIE I, II and III.* In D Sleeman and JS Brown (eds), *Intelligent Tutoring Systems*, Academic Press, pp 227-282, 1982.

[Buchanan and Feigenbaum, 1978] Buchanan BG and Feigenbaum E. *DENDRAL and Meta-DENDRAL: Their Applications Dimension.* Artificial Intelligence, Vol 11, pp 5-24, 1978.

[Buchanan and Shortliffe, 1984] Buchanan BG and Shortliffe EH. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley, 1984.

[Campbell et al, 1981] Campbell AN, Hollister VF, Duda RO and Hart PE. *Recognition of a hidden mineral deposit by an Artificial Intelligence Program.* Science, Vol 217, pp 927-929, September 3, 1982.

[Carbonell, 1970] Carbonell JR. *AI in CAI: An Artificial Intelligence Approach to Computer-Aided Instruction.* IEEE Transactions on Man-Machine Systems, Vol 11, No 4, pp 190-202, 1970.

[Carhart, 1979] Carhart RE. *CONGEN: An Expert System Aiding the Structural Chemist.* In [Michie, 1979], pp 65-82.

[Chandrasekar and Ramani, 1989] Chandrasekar R and Ramani S. *Interactive Communication of Sentential Structure and Content: An Alternative Approach to Man-machine Communication.* International Journal of Man Machine Studies, Vol 30, No 2, February 1989.

[Chang and Lee, 1973] Chang CL and Lee RCT. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, 1973.

[Charniak and McDermott, 1985] Charniak E and McDermott D. *Introduction to Artificial Intelligence.* Addison Wesley, 1985.

[Clancey, 1983] Clancey WJ. *The Epistemology of a Rule-Based Expert System: A Framework for Explanation.* Artificial Intelligence, Vol 20, No 3, pp 215-251, 1983.

[Clancey, 1985] Clancey WJ. *Heuristic Classification.* Artificial Intelligence, Vol 27, pp 289-350, 1985.

[Clancey, 1987] Clancey WJ. *Knowledge Based Tutoring: The GUIDON Program.* MIT Press, 1987.

[Clancey, 1992] Clancey WJ. *Model Construction Operators.* Artificial Intelligence, Vol 53, No 1, pp 1-115, 1992.

[Clancey and Letsinger, 1981] Clancey WJ and Letsinger R. *NEOMYCIN: Reconfiguring a Rule-based Expert System for Application to Teaching.* Proceedings of IJCAI-81, pp 829-836, 1981.

[Clancey and Shortliffe, 1984] Clancey WJ and Shortliffe EH. *Readings in Medical Artificial Intelligence: The First Decade.* Addison Wesley, 1984.

[Clocksin and Mellish, 1989] Clocksin W and Mellish C. *Programming in PROLOG, Third Edition.* Springer-Verlag, 1989.

[Cohen, 1985] Cohen PR. *Heuristic Reasoning about Uncertainty: An Artificial Intelligence Approach.* Pitman, 1985.

[Collins and Quillian, 1972] Collins AM and Quillian MR. *Experiments on Semantic Memory and Language Comprehension.* In Gregg LW (ed), *Cognition in Learning and Memory.* Wiley, pp 117-137, 1972.

[Cooper and Wogrin, 1988] Cooper TA and Wogrin N. *Rule-based Programming with OPS5.* Morgan Kaufmann, 1988.

[Cortes-Rello and Golshani, 1990] Cortes-Rello E and Golshani F. *Uncertain Reasoning using the Dempster-Shafer Method: An Application in Forecasting and Marketing Management.* Expert Systems, February 1990, Vol 7, No 1, pp 9-18, 1990.

[Dague et al, 1987] Dague P, Raiman O and Deves P. *Troubleshooting: When Modelling is the Difficulty.* Proceedings of AAAI-87, pp 600-605, 1987.

[Davis, 1979] Davis R. *Interactive Transfer of Expertise: Acquisition of New Inference Rules.* Artificial Intelligence, Vol 12, pp 121-157, 1979. Also in [Gupta and Prasad, 1988], pp 243-261.

[Davis and Hamscher, 1988] Davis R and Hamscher W. *Model-based Reasoning: Troubleshooting.* In Howard E Shrobe and AAAI (eds), *Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence*, Morgan Kaufmann, pp 297-346, 1988.

[Davis and King, 1984] Davis R and King JJ. *The Origin of Rule-Based Systems in Artificial Intelligence.* In [Buchanan and Shortliffe, 1984], pp 20-52.

[Davis et al, 1977] Davis R, Buchanan BG and Shortliffe EH. *Production Rules as a Representation for a Knowledge-Based Consultation Program.* Artificial Intelligence, Vol 8, pp 15-45, 1977. Also in [Clancey and Shortliffe, 1984], pp 98-130.

[deKleer, 1986] deKleer J. *An Assumption Based TMS.* Artificial Intelligence, Vol 28, pp 127-162, 1986.

[Doyle, 1979] Doyle J. *A Truth Maintenance System.* Artificial Intelligence, Vol 12, pp 231-272, 1979.

[Duda et al, 1978] Duda RO, Hart PE and Nilsson NJ. *Semantic Network Application in Rule-based Inference Systems.* In [Waterman and Hayes-Roth, 1978], pp 203-221.

[Duda et al, 1979] Duda R, Gaschnig J and Hart P. *Model Design in the Prospector Consultant System for Mineral Exploration.* In [Michie, 1979], pp 153-167.

[Dumas, 1988] Dumas, JS. *Designing User Interfaces for Software.* Prentice-Hall, 1988.

[Feigenbaum and Feldman, 1963] Feigenbaum EA and Feldman J (eds). *Computers and Thought.* McGraw-Hill, 1963.

[Feigenbaum et al, 1988] Feigenbaum EA, McCorduck P and Nii HP. *The Rise of the Expert Company.* Vintage Books, 1988.

[Feldman and Rovner, 1969] Feldman JA and Rovner PD. *An Algol-Based Associate Language.* Communications of the ACM, Vol 12, No 8, pp 439-449, August 1969.

[Findler, 1979] Findler NV (ed). *Associative Networks: Representation and Use of Knowledge by Computers.* Academic Press, 1979.

[Finin, 1986] Finin Tim. *PFL - A Pedagogical Frame Language.* Technical Report MS-CIS-86-72, LINC LAB 35, Computer and Information Science Department, University of Pennsylvania, Philadelphia, PA, USA, June 1986, revised November 1986.

[Forgy, 1982] Forgy CL. *Rete: A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem.* Artificial Intelligence, Vol 19, pp 17-37, 1982.

[Genesereth, 1984] Genesereth MR. *The Use of Design Descriptions in Automated Diagnosis.* Artificial Intelligence, Vol 24, pp 411-436, 1984.

[Genesereth and Nilsson, 1987] Genesereth MR and Nilsson NJ. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann, 1987.

[Ginsberg, 1988] Ginsberg A. *Automatic Refinement of Expert System Knowledge Bases.* Morgan Kaufmann, 1988.

[Ginsberg et al, 1987] Ginsberg A, Weiss S and Politakis P. *SEEK2: A Generalized Approach to Automatic Knowledge Base Refinement.* Proceedings of IJCAI-87, Morgan Kaufmann, pp 367-374, 1987.

[Gordon and Shortliffe, 1984] Gordon J and Shortliffe EH. *The Dempster-Shafer Theory of Evidence.* In [Buchanan and Shortliffe, 1984], pp 272-292.

[Gordon and Shortliffe, 1985] Gordon J and Shortliffe EH. *A Method for Managing Evidential Reasoning in Hierarchical Hypothesis Spaces.* Artificial Intelligence, Vol 26, pp 323-358, 1985.

[Grimson and Patil, 1987] Grimson WEL and Patil RS (eds). *AI in the 1980s and Beyond: An MIT Survey.* MIT Press, 1987.

[Gupta and Prasad, 1988] Gupta A and Prasad BE (eds). *Principles of Expert Systems.* IEEE Press, 1988.

[Harmon and King, 1985] Harmon P and King D. *Expert Systems: Artificial Intelligence in Business.* John Wiley and Sons, 1985.

[Hayes, 1979] Hayes PJ. *The Logic of Frames.* In Metzing D (ed), *Frame Conceptions and Text Understanding*, Walter de Gruyter and Co, pp 46-61, 1979. Also Chapter 14 in [Brachman and Levesque, 1985], pp 287-295.

[Hendrix, 1975] Hendrix GG. *Expanding the Utility of Semantic Networks through Partitioning.* Proceedings of IJCAI-75, pp 115-121, 1975.

[Jackson et al, 1989] Jackson P, Reichgelt H and Harmelen F (eds). *Logic-based Knowledge Representation.* MIT Press, 1989.

[Julian, 1983] Julian DG. *Cardiology.* Fourth Edition, English Language Book Society and Bailliere Tindall, London, 1983.

[Kowalski, 1974] Kowalski R. *Predicate Logic as a Programming Language.* Proceedings of IFIP-74, North-Holland, pp 569-574, 1974.

[Kowalski, 1979a] Kowalski R. *Logic for Problem Solving.* North-Holland, 1979.

[Kowalski, 1979b] Kowalski R. *Algorithm = Logic + Control.* Communications of the Association for Computing Machinery, Vol 22, pp 424-436, 1979.

[Kunz et al, 1978] Kunz JC, Fallat RJ, McClung DH, Osborn JJ, Votteri RA, Nii HP, Aikins JS, Fagan LM and Feigenbaum EA. *A Physiological Rulebased System for Pulmonary Function Test Results.* Technical Report of the Heuristic Programming Project, HPP-78-19, Stanford University, 1978.

[Levesque and Brachman, 1984] Levesque HJ and Brachman RJ. *A Fundamental Tradeoff in Knowledge Representation and Reasoning (revised version).* Proceedings of CSCSI-84, London, Ontario, pp 141-152, 1984. Also Chapter 4 in [Brachman and Levesque, 1985], pp 41-70.

[Lighthill, 1973] Lighthill J. *Artificial Intelligence: A General Survey.* In the Proceedings of Artificial Intelligence: A Paper Symposium, conducted by the Science Research Council in April, 1973.

[Lloyd, 1984] Lloyd J. *Foundations of Logic Programming.* Springer-Verlag, 1984.

[McCarthy, 1980] McCarthy J. *Circumscription - A Form of Non-monotonic Reasoning.* Artificial Intelligence, Vol 13, pp 27-39, 1980.

[McDermott, 1981] McDermott J. *R1: The Formative Years.* AI Magazine, Vol 2, No 2, pp 21-29, Summer 1981.

[McDermott, 1982] McDermott J. *R1: A Rule-based Configurer of Computer Systems.* Artificial Intelligence, Vol 19, pp 39-88, 1982.

[McDermott, 1984] McDermott J. *R1 Revisited: Four Years in the Trenches.* AI Magazine, pp 21-32, Fall 1984.

[McDermott and Doyle, 1980] McDermott D and Doyle J. *Non-monotonic Logic I.* Artificial Intelligence, Vol 13, pp 41-72, 1980.

[Michalski, 1983] Michalski RS. *A Theory and Methodology of Inductive Reasoning.* Artificial Intelligence, Vol 20, pp 111-161, 1983. Also in [Michalski et al, 1983], pp 83-184 and in [Shavlik and Dietterich, 1990], pp 70-95.

[Michalski et al, 1983] Michalski RS, Carbonell JG and Mitchell TM. *Machine Learning*. Tioga Publishers, 1983.

[Michalski et al, 1986] Michalski RS, Carbonell JG and Mitchell TM. *Machine Learning, Vol II*. Tioga Publishers, 1986.

[Michie, 1979] Michie D (ed). *Expert Systems in the Micro-electronic Age*. Edinburgh University Press, 1979.

[Minsky, 1975] Minsky M. *A Framework for Representing Knowledge*. In [Winston, 1975], pp 211-277.

[Minsky, 1981] Minsky M. *A Framework for Representing Knowledge (revised version)*. In J Haugeland (ed), *Mind Design*, MIT Press, pp 95-128, 1981. Also Chapter 12 in [Brachman and Levesque, 1985], pp 245-262.

[Moore and Swartout, 1988] Moore JD and Swartout WR. *Explanation in Expert Systems: A Survey*. ISI Technical Report, University of Southern California, ISI/RR-88-228, December 1988.

[NCST, 1993] NCST. *Vidwan - An Expert System Shell: User Manual*. National Centre for Software Technology, Bombay, 1993.

[Negoita, 1985] Negoita CV. *Expert Systems and Fuzzy Systems*. Benjamin/Cummings, 1985.

[Newell and Simon, 1972] Newell A and Simon HA. *Human Problem Solving*. Prentice Hall, 1972.

[Nguyen, 1987] Nguyen TA. *Verifying Consistency of Production Systems*. Proceedings of the IEEE Third Conference on Artificial Intelligence Applications, pp 4-8, 1987.

[Nievergelt and Weydert, 1980] Nievergelt J and Weydert J. *Sites, Modes and Trails: Telling the User of an Interactive System Where He is, What He Can do, and How to Get Places*. In RA Guedj (ed), *Methodology of Interaction*, North Holland, pp 327-338, 1980.

[Nilsson, 1980] Nilsson NJ. *Principles of Artificial Intelligence*. Springer Verlag, 1980.

[Nyberg, 1988] Eric H Nyberg 3rd. FRAMEKIT *User's Guide Version 2.0*. CMU-CMT-Memo, Center for Machine Translation, Carnegie Mellon University, Pittsburgh, PA, USA, 1988.

[Palekar, 1991] Palekar BB. *Tax Planning for You and Me*. Popular Prakashan Pvt Ltd, 1991

[Perkins et al, 1989] Perkins WA, Laffey TJ, Pecora D and Nguyen TA. *Knowledge Base Verification*. In G Guida and C Tasso (eds), *Topics in Expert System Design*, Elsevier Science, pp 353-376, 1989.

[Politakis, 1985] Politakis P. *Empirical Analysis of Expert Systems*. Morgan Kaufmann, 1985.

[Politakis and Weiss, 1984] Politakis P and Weiss SM. *Using Empirical Analysis to Refine Expert System Knowledge Bases*. Artificial Intelligence, Vol 22, pp 23-48, 1984.

[Post, 1943] Post E. *Formal Reductions of the General Combinatorial Problem*. American Journal of Mathematics, Vol 65, pp 197-268, 1943.

[Quillian, 1967] Quillian MR. *Word Concepts: A Theory and Simulation of Some Basic Semantic Capabilities.* Behavioural Science, Vol 12, pp 410-430, 1967. Also Chapter 6 in [Brachman and Levesque, 1985], pp 97-118.

[Quillian, 1968] Quillian MR. *Semantic Memory.* In M Minsky (ed), *Semantic Information Processing*, MIT Press, pp 227-270, 1968.

[Quinlan, 1979] Quinlan JR. *Discovering Rules from a Large Collection of Examples: a Case Study.* In [Michie, 1979], pp 168-201.

[Quinlan, 1986] Quinlan JR. *Induction of Decision Trees.* In [Michalski et al, 1983], pp 81-106. Also in [Shavlik and Dietterich, 1990], pp 57-69.

[Rabinowitz et al, 1991] Rabinowitz H, Flamholz J, Wolin E and Euchner J. *NYNEX MAX: A Telephoning Trouble Screening Expert.* In Smith R and Scott C (eds), *Innovative Applications of Artificial Intelligence 3*, The AAAI Press, 1991.

[Ramani, Chandrasekar and Anjaneyulu, 1990] Ramani S, Chandrasekar R and Anjaneyulu KSR (eds). *Knowledge-Based Computer Systems.* Narosa Publishing House, 1990. Also published as Vol 444, *Lecture Notes in Computer Science*, Springer-Verlag, 1990.

[Ramchandar, 1992] Ramchandar K. CONTAX*: An Expert Consultant for Income Tax Planning.* MCA Thesis, NCST, 1992.

[Rauch-Hindin, 1988] Rauch-Hindin WB. *A Guide to Commercial Artificial Intelligence: Fundamentals and Real World Applications.* Prentice Hall, 1988.

[Register and Rewari, 1991] Register MS and Rewari A. *CANASTA: The Crash Analysis Troubleshooting Assistant.* In Smith R and Scott C (eds), *Innovative Applications of Artificial Intelligence 3*, The AAAI Press, 1991.

[Reiter, 1980] Reiter R. *A Logic for Default Reasoning.* Artificial Intelligence, Vol 13, pp 81-132, 1980.

[Reiter, 1987] Reiter R. *A Theory of Diagnosis from First Principles.* Artificial Intelligence, Vol 32, pp 57-95, 1987.

[Rich and Knight, 1991] Rich E and Knight K. *Artificial Intelligence.* McGraw-Hill, 1991.

[Richards, 1989] Richards T. *Clausal Form Logic: An Introduction to the Logic of Computer Reasoning.* Addison-Wesley, 1989.

[Roberts and Goldstein, 1977] Roberts RB and Goldstein IP. *The FRL Manual.* AI Memo No. 409, AI Laboratory, MIT, 1977.

[Robinson, 1965] Robinson JA. *A Machine-Oriented Logic Based on the Resolution Principle.* Journal of the Association for Computing Machinery, Vol 12, pp 23-41, 1965.

[Rosenbloom et al, 1991] Rosenbloom PS, Laird JE, Newell A and McCarl R. *A Preliminary Analysis of the Soar Architecture as a Basis for General Intelligence.* Artificial Intelligence, Vol 47, No 1-3, pp 289-325, 1991.

[Rumelhart et al, 1972] Rumelhart DE, Lindsay PH and Norman DA. *A Process Model for Long-term Memory.* In Tulving E and Donaldson W (eds), *Organization of Memory*, Academic Press, pp 197-246, 1972.

[Sasikumar, 1989] Sasikumar M. *From Symbolic Logic to Prolog.* Internal Note, NCST, 1989.

[Sasikumar and Ramchandar, 1992] Sasikumar M and Ramchandar K. *Expert Systems in Finance.* Internal Report, NCST, 1992.

[Sasikumar et al, 1988] Sasikumar M, Muthu Raman S and Ramani S. *Tools and Techniques for the Refinement of Rulebased Expert Systems.* Proceedings of National Conference on Knowledge Based Computer Systems, Bangalore, pp. 10-13, June 10-11, 1988.

[Shafer, 1976] Shafer G. *A Mathematical Theory of Evidence.* Princeton University Press, 1976.

[Schank and Abelson, 1977] Schank RC and Abelson RP. *Scripts, Plans, Goals and Understanding.* Lawrence Erlbaum, 1977.

[Shafer and Pearl, 1990] Shafer G and Pearl J (eds). *Readings in Uncertain Reasoning.* Morgan Kaufmann, 1990.

[Shanbag, 1991] Shanbag AN. *In the Wonderland of Investments.* Popular Prakashan Pvt Ltd, 1991.

[Shapiro, 1979] Shapiro SC. *The SNePS Semantic Network Processing System.* In [Findler, 1979], pp 179-203.

[Shapiro, 1987] Shapiro AD. *Structured Induction in Expert Systems.* Addison-Wesley, 1987.

[Shavlik and Dietterich, 1990] Shavlik JW and Dietterich TG (eds). *Readings in Machine Learning.* Morgan Kaufmann, 1990.

[Shneiderman, 1987] Shneiderman B. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley, 1987.

[Shortliffe, 1976] Shortliffe EH. *Computer-Based Medical Consultations: MYCIN.* Elsevier, 1976.

[Shortliffe and Buchanan, 1975] Shortliffe EH and Buchanan BG. *A Model of Inexact Reasoning in Medicine.* Mathematical Biosciences, Vol 23, pp 351-379, 1975. Also in [Buchanan and Shortliffe, 1984], pp 233-262.

[Siegfried and Wright, 1986] Siegfried EM and Wright JR. *ACE: Taking an Expert System from Prototype to Product.* In Bernold T (ed), *Expert Systems and Knowledge Engineering*, North Holland, pp 121-131, 1986.

[Singhania, 1992] Singhania VK. *Students Guide to Income Tax.* Taxmann Publications Pvt Ltd, 1992.

[Simmons and Bruce, 1971] Simmons RF and Bruce BC. *Some Relations between Predicate Calculus and Semantic Net Representations of Discourse.* Proceedings of IJCAI-71, pp 524-529, 1971.

[Sowa, 1984] Sowa JF. *Conceptual Structures: Information Processing in Minds and Machines.* Addison-Wesley, 1984.

[Sowa, 1991] Sowa JF (ed). *Principles of Semantic Networks: Explorations in the Representation of Knowledge.* Morgan Kaufmann, 1991.

[Sterling, 1990] Sterling L (ed). *The Practice of Prolog*. MIT Press, 1990.

[Swartout, 1983] Swartout WR. *XPLAIN: A System for Creating and Explaining Expert Consulting Programs.* Artificial Intelligence, Vol 21, pp 285-325, 1983.

[Turing, 1950] Turing A. *Computing Machinery and Intelligence.* Mind, Vol 59, pp 433-460, 1950. Also in [Feigenbaum and Feldman, 1963].

[van Melle et al, 1984] van Melle W, Shortliffe EH and Buchanan B. *EMYCIN: A Knowledge Engineers Tool for Constructing Rule-based Expert Systems.* In [Buchanan and Shortliffe, 1984], pp 302-313, 1984.

[Versonder et al, 1983] Versonder GT, Stolfo SJ and Zielinski JE. *ACE: An Expert System for Telephone Cable Maintenance.* Proceedings of IJCAI-83, pp 116-121, 1983.

[Wallwork and Stepney, 1987] Wallwork J and Stepney R. *Heart Disease, What it is and How it's Treated.* Basil Blackwell, 1987.

[Warren, 1983] Warren DHD. *Applied Logic – Its Use and Implementation as a Programming Tool.* Technical Note 290, SRI International, 1983.

[Waterman and Hayes-Roth, 1978] Waterman DA and Hayes-Roth F (eds). *Pattern Directed Inference Systems.* Academic Press, 1978.

[Whitehead and Roach, 1990] Whitehead JD and Roach JW. *HOIST: A Second Generation Expert System Based on Qualitative Physics.* AI Magazine, Vol 11, No 3, pp 108-119, Fall 1990.

[Winograd, 1975] Winograd T. *Frame Representations and the Declarative/Procedural Controversy.* In [Bobrow and Collins, 1975], pp 185-210.

[Winston, 1975] Winston PH. *Learning Structural Descriptions from Examples.* In PH Winston (ed), *The Psychology of Computer Vision*, McGraw-Hill, 1975. Also Chapter 8 in [Brachman and Levesque, 1985], pp 141-168.

[Woods, 1975] Woods WA. *What's in a Link: Foundations for Semantic Networks.* In [Bobrow and Collins, 1975], pp 35-82. Also Chapter 11 in [Brachman and Levesque, 1985], pp 217-241.

[Zadeh, 1984] Zadeh LA. *Review of Shafer's "A Mathematical Theory of Evidence".* AI Magazine, pp 81-83, Fall 1984.

[Zadeh, 1986] Zadeh LA. *A Simple View of Dempster-Shafer Theory of Evidence and its Implication for the Rule of Combination.* AI Magazine, pp 85-90, 1986.

[Zadeh, 1988] LA Zadeh. *Fuzzy Logic.* IEEE Computer, Vol 21, No 4, pp 83-93, April 1988.