What Are Expert Systems?

1.1 The nature of expertise
1.2 The characteristics of an expert system
1.3 Fundamental topics in expert systems
1.4 Summary and chapter plan
Bibliographical notes
Study suggestions

You are having a bad day with your personal computer. You have just upgraded your favorite word processor, and now when you click the icon you get an error message, such as

"Call to Undefined Link."

Like most error messages, this is about as helpful as a fortune cookie from Mars. You blow the whole directory away and reinstall, to no avail. You tinker with various initialization files, but nothing works.

Finally, you break down and call customer support. Your luck changes and you get someone who knows what they are talking about. He tells you to blow away half a dozen old dynamic link libraries in a system directory and reinstall. You do it, it works, and your blood pressure returns to normal.

Whatever expertise may be in this domain, it is clear that the technical support person has it and you don’t. You may have a Ph.D. in Computer Science, you may be a wiz programmer, but you couldn’t do their job unless you underwent the right training and somehow acquired their troubleshooting experience. So expertise isn’t just possessing knowledge or having qualifications; it is a highly specialized set of skills that have been honed in a particular situation for a specific purpose. As such, being an expert is quite distinct from having an education.

1.1 The nature of expertise

That being said, under what conditions would you feel happy in calling a computer program an expert?
1.2 The characteristics of an expert system

An expert system can be distinguished from a more conventional applications program in that:

- It simulates human reasoning about a problem domain, rather than simulating the domain itself. This distinguishes expert systems from more familiar programs that involve mathematical modeling or computer animation. This is not to say that the program is a faithful psychological model of the expert, merely that the focus is upon emulating an expert's problem solving abilities, that is, performing some portion of the relevant tasks as well as, or better than, the expert.
- It performs reasoning over representations of human knowledge, in addition to doing numerical calculations or data retrieval. The knowledge in the program is normally expressed in some special-purpose language and kept separate from the code that performs the reasoning. These distinct program modules are referred to as the knowledge base and the inference engine, respectively.
- It solves problems by heuristic or approximate methods which, unlike algorithmic solutions, are not guaranteed to succeed. A heuristic is essentially a rule of thumb which encodes a piece of knowledge about how to solve problems in some domain. Such methods are approximate in the sense that (i) they do not require perfect data and (ii) the solutions derived by the system may be proposed with varying degrees of certainty.

An expert system differs from other kinds of artificial intelligence program in that:

- It deals with subject matter of realistic complexity that normally requires a considerable amount of human expertise. Many AI programs are really research vehicles, and may therefore focus upon abstract mathematical problems or simplified versions of real problems (sometimes called ‘toy’ problems) in order to gain insights or refine techniques. Expert systems, on the other hand, solve problems of genuine scientific or commercial interest.
- It must exhibit high performance in terms of speed and reliability in order to be a useful tool. AI research vehicles may not run very fast, and may well contain bugs: they are programs, not supported software. But an expert system must propose solutions in a reasonable time and be right most of the time, that is, at least as often as a human expert.
- It must be capable of explaining and justifying solutions or recommendations in order to convince the user that its reasoning is in fact correct. Research programs are typically only run by their creators, or by other personnel in similar laboratories. An expert system will be run by a wider range of users, and should therefore be designed in such a way that its workings are rather more transparent.

The term knowledge-based system is sometimes used as a synonym for ‘expert system’ though, strictly speaking, the former is more general. A knowledge-based system is any system which performs a task by applying rules of thumb to a symbolic representation of knowledge, instead of employing mostly algorithmic or statistical methods. Thus a program capable of discussing the weather would be a knowledge-based system, even if that program did not embody any expertise in meteorology, but an expert
system in the domain of meteorology ought to be able to provide us with weather forecasts.

In summary, expert systems encode the domain-dependent knowledge of everyday practitioners in some field, and use this knowledge to solve problems, instead of using comparatively domain-independent methods derived from computer science or mathematics. The process of constructing an expert system is often called knowledge engineering, and is considered to be 'applied artificial intelligence' (Feigenbaum, 1977). We shall further develop the distinction between the knowledge engineering approach and more conventional computer science approaches to problem solving in Chapters 2 and 3.

The rest of this chapter has the following plan. Four fundamental topics are identified which are intimately related to the theory and practice of expert systems development, and these are briefly discussed with a view to introducing some terminology and giving the reader an overview. The final section poses the question 'What is the state of the art?' and provides a chapter plan for the rest of the book.

1.3 Fundamental topics in expert systems

Given that expert systems research has grown out of more general concerns in artificial intelligence, it is not surprising that it maintains strong intellectual links with related topics in its parent discipline. Some of these links are outlined in the following sections, with references to the general literature. There are also forward references to chapters of this book which have a bearing on the various topics.

1.3.1 Acquiring knowledge

Buchanan et al. (1983) define knowledge acquisition as follows.

(Knowledge acquisition is) 'the transfer and transformation of potential problem-solving expertise from some knowledge source to a program'.

This transfer is usually accomplished by a series of lengthy and intensive interviews between a knowledge engineer, who is normally a computer specialist, and a domain expert who is able to articulate his expertise to some degree. It is estimated that this form of labor produces between two and five units of knowledge (for example, rules of thumb) per day. This rather low output has led researchers to look upon knowledge acquisition as 'the bottleneck problem' of expert systems applications (Feigenbaum, 1977).

There are a number of reasons why productivity is typically so poor; here are some of them.

- Specialist fields have their own jargon, and it is often difficult for experts to communicate their knowledge in everyday language (see Box 1.1). Analyzing the concepts behind the jargon is rarely straightforward, since these concepts need not admit of precise mathematical or logical definition. For example, a military strategist may speak of the 'aggressive posture' of a foreign power without being able to specify exactly what distinguishes such a posture from a non-threatening one.

- The facts and principles underlying many domains of interest cannot be characterized precisely in terms of a mathematical theory or a deterministic model whose properties are well understood. Thus a financial expert may know that certain events cause the stock market to go up or down, but the exact mechanisms that mediate these effects, and the magnitude of the effects themselves, cannot be identified or predicted with certainty. Statistical models may enable us to make rather general, long-term predictions, but they do not normally sanction specific courses of action in the short term.

- Experts need to know more than the mere facts or principles of a domain in order to solve problems. For example, they usually know which kinds of information are relevant to which kinds of judgment, how reliable different information sources are, and how to make hard problems easier by splitting them into subproblems which can be solved more or less independently. Eliciting this kind of knowledge, which is normally based on personal experience rather than formal training, is much more difficult than eliciting either particular facts or general principles.

- Human expertise, even in a relatively narrow domain, is often set in a broader context that involves a good deal of common sense knowledge about the everyday world. Consider legal experts involved in litigation. It is difficult to delineate the amount and nature of general knowledge needed to deal with an arbitrary case.

---

**Box 1.1 Forgotten Passwords**

If you have a memory like mine, you probably forget your passwords on various machines quite frequently. How does your system administrator, let's call him Sam, reset your password? Our budding knowledge engineer, Ken, tries to find out.

Ken: Well, if it's a YP password, I first log on as root on the YP master.

Sam: It's the diskfull master that contains a database of network information.

Ken: Diskfull meaning? It has the OS installed on local disk.

Sam: Ah, (Scrubbles furiously) So you log on... Ken: As root. Then I edit the password datafile, remove the encrypted entry, and make the new password map.

Ken: Password map. (Attempting humor) What happens if you forget your password?

Sam: On a diskfull system, I could reboot to single user mode, or I could load MINIROOT so I can edit /etc/passwd. Or I could relode the whole system, which'd rather not do. Root passwords aren't usually included in YP. On a diskless client I could use the passwd command.

Ken: Oh, (Sorry he asked)

Poor Ken is struggling. He might have had an easier time if he had glanced at the System Administrator's Guide prior to the interview to get some basic terminology.
Dissatisfaction with the interview method has led some researchers to try to automate the process of knowledge acquisition. One area of research concerns automated knowledge elicitation, in which an expert's knowledge is transferred to a computer program as a side effect of a person–machine dialogue (see Chapter 10 et seq.). Other researchers have looked to the subfield of AI known as machine learning for a solution to the bottleneck problem. The idea is that a computing system could perhaps learn to solve problems in the same way that humans do, that is to say, by example (see Chapter 20).

1.3.2 Representing knowledge

Knowledge representation is a substantial subfield in its own right, which shares many concerns with both formal philosophy and cognitive psychology. It is concerned with the ways in which information might be stored and associated in the human brain, usually from a logical, rather than a biological, perspective. In other words, it is not typically concerned with the physical details of how knowledge is encoded, but rather with what the overall conceptual scheme might look like.

\[
\text{Syntax and Semantics in the Family}
\]

A basic part of knowledge representation, which is so obvious that it is often not stressed, is that a representation should somehow "standardize" the syntactic variety of English. Thus the sentences

'\text{Sam is the father of Bill.}'

'\text{Sam is Bill's father.}'

'\text{Bill's father is Sam.}'

'\text{Bill has Sam for a father.}'

all have the same fundamental meaning (semantic content), and should therefore be rendered in the same way. In a representation, we try to relate form and meaning more simply than in English, in that expressions with the same or similar meaning should look the same or similar. Thus the above sentences might be mapped to the expression

\[
\text{father(sam, bill)}.
\]

The semantics of this expression must specify (amongst other things) that the first name stands for the parent and the second name stands for the child, and not vice versa.

We can also see that the sentences

'\text{Sam is the father of Jill.}'

'\text{Bill has Sam for a father.}'

share meaning in a manner that is more evident when we render them as

\[
\text{father(sam, bill)}; \\
\text{father(jill, sam)}.
\]

We shall say more about syntax and semantics in Chapters 3 and 8.

In the 1970s, knowledge representation research attempted to address such questions as how human memory works, proposing theories of remining, recognition, and recall. Some of the resultant theories led to computer programs which tried to simulate different ways in which concepts might be associated, so that a computer application would somehow always be able to find the right piece of knowledge at the right time when solving some problem. Over time, the psychological veracity of these theories became less important than their utility as vehicles for experimenting with new data and control structures, at least from an AI point of view.

Knowledge representation in the large has always been, and is likely to remain, a controversial topic. Philosophers and psychologists have sometimes been shocked by the hubris of AI researchers who talk rather glibly about human knowledge in a jargon that freely mixes terminology from logic, linguistics, philosophy, psychology, and computer science. On the other hand, the computer metaphor has provided a novel means of asking, and occasionally answering, difficult questions which have languished for centuries in the realm of metaphysics.

In the world of expert systems, knowledge representation is mostly concerned with finding ways in which large bodies of useful information can be formally described for the purposes of symbolic computation. Formally described means rendered in some unambiguous language or notation which has a well-defined syntax governing the form of expressions in the language, and a well-defined semantics which reveals the meaning of such expressions by virtue of their form. We can put off a discussion of syntax and semantics until Chapter 3.

Symbolic computation means non-numeric computations in which the symbols and symbol structures can be construed as standing for (representing) various concepts and relationships between them. We shall likewise put off a discussion of symbolic computation until Chapter 4. But see Box 1.2 for an example of symbolic representation.

AI researchers have expended a good deal of effort in constructing representation languages, that is, computer languages that are oriented towards organizing descriptions of objects and ideas, rather than stating sequences of instructions or storing simple data elements. The main criteria for assessing a representation of knowledge are logical adequacy, heuristic power, and notational convenience; these terms deserve some explanation.

- **Logical adequacy** means that the representation should be capable of making all the distinctions that you want to make. For example, it is not possible to represent the idea that every drug has some undesirable side effect unless you are able to distinguish between the designation of a particular drug and a particular side effect (for example, aspirin aggravates ulcers) and the more general statement to the effect that: 'for any drug, there is an undesirable side effect associated with it'.

- **Heuristic power** means that as well as having an expressive representation language, there must be some way of using representations so constructed and interpreted to solve problems. It is often the case that the more expressive the language, in terms of the number of semantic distinctions that it can make, the more difficult it is to control the drawing of inferences during problem solving. Many of the formalisms that have found favor with practitioners may seem quite restricted in terms of their powers of expression when compared with English or even
standard logic. Yet they frequently gain in heuristic power as a consequence; that is, it is relatively easy to bring the right knowledge to bear at the right time. Knowing which areas of knowledge are most relevant to which problems is one of the things that distinguishes the expert from the amateur or the merely well-read.

- **Notational convenience** is a virtue because most expert systems applications require the encoding of substantial amounts of knowledge, and this task will not be an enviable one if the conventions of the representation language are too complicated. The resulting expressions should be relatively easy to write and to read, and it should be possible to understand their meaning without knowing how the computer will actually interpret them. The term **declarative** is often used to describe code which is essentially prescriptive and can therefore be understood without knowing what states a real or virtual machine will go through at execution time.

Several conventions for coding knowledge have been suggested, including **production rules** (Davis and King, 1977), **structured objects** (Findler, 1979) and **logic programs** (Kowalski, 1979). Most expert systems use one or more of these formalisms, and their pros and cons are still a source of controversy among theoreticians. A number of such formalisms are critically reviewed in Chapters 5–8, while software tools for constructing such representations are described in Chapters 17, 18 and 19.

Most of the code examples in this book use the **CLIPS** language, which combines production rules with structured objects. The appendix provides a reasonably thorough introduction to the main concepts and constructs of **CLIPS**, with many code examples. There the reader will find a non-trivial program which exercises most of the interesting features of **CLIPS** and demonstrates many of the AI techniques discussed in Chapters 1–3.

### 1.3.3 Controlling reasoning

Expert systems design involves paying close attention to the details of how knowledge is accessed and applied during the search for a solution (Davis, 1980a). Knowing what one knows, and knowing when and how to use it, is an important part of expertise; this is usually termed **metaknowledge**, that is, knowledge about knowledge. Solving non-trivial problems implies a certain level of **planning and control** when choosing what questions to ask, what tests to perform, and so on.

Different strategies for bringing domain-specific knowledge to bear will generally have marked effects upon the performance characteristics of programs. They determine the manner in which a program **searches** for a solution in some space of alternatives (see Chapters 2 and 3). It will not normally be the case that the data given to a knowledge-based program will be sufficient for the program to deduce exactly where it should look in this space.

Most knowledge representation formalisms can be employed under a variety of **control regimes** (see Box 1.3), and expert systems researchers are continuing to experiment in this area. The systems reviewed in the later chapters have been specially chosen to illustrate the many different ways in which the problem of control can be tackled. Each has something to offer the student of expert systems research and development.

### Box 1.3: Automotive Aggravation

Imagine that your car is difficult to start, and when running exhibits loss of power. These symptoms are not, in themselves, sufficient for you to decide whether you should look for a fault in the electrical system or the fuel system of your car. Yet your knowledge of cars might tell you that it is worth running some additional checks before calling a mechanic. Perhaps the mixture is wrong, so look at the exhaust smoke and the coating on the spark plugs. Maybe the distributor is faulty, so see if the cap is damaged in some way. These rather specific heuristics are not guaranteed to locate the fault, but with luck they may take you to the heart of the problem more quickly than running an exhaustive set of routine checks over the car’s components.

Even if you are mystified by the symptoms of your sick vehicle, you probably know enough to perform global checks before very specific checks, for example, to see if there is a strong spark at the plugs (which would tend to rule out an electrical fault) before testing the battery for flatness. Even in the absence of specific heuristics, the more methodical your procedures, the greater the chance that you will find the fault quickly. The general heuristic that says

- **test whole modules before testing their components**
- **could form part of a control regime: a strategy for applying knowledge in some systematic way. Another heuristic might be**
- **replace cheaper parts before replacing more expensive ones.**

These two heuristics may give contradictory advice in some cases, and so one might have to be more dominant than the other if they are to coexist in the same control regime.

### 1.3.4 Explaining solutions

The whole question of how to help a user understand the structure and function of some complex piece of software relates to the comparatively new field of **human-computer interaction**, which is emerging from an intersection of AI, engineering, psychology, and ergonomics. The contribution of expert systems researchers to date has been to place a high priority upon the accountability of consultation programs, and to show how explanations of program behavior can be systematically related to the chains of reasoning employed by such systems. Ongoing contributions include attempts to separate out the different kinds of knowledge implicit in expert performance, and attempts to make explicit and accessible the design decisions associated with the specification of consultation programs.

Explanations of expert system behavior are important for a number of reasons:

- **users** of the system need to satisfy themselves that the program’s conclusions are basically correct for their particular case;
- **knowledge engineers** need some way to satisfy themselves that knowledge is being applied properly even as the prototype is being built;
- **domain experts** need to see a trace of the way in which their knowledge is being applied in order to judge whether knowledge elicitation is proceeding successfully;
- **programmers** who maintain, debug and extend knowledge-based programs must have some window onto the program’s behavior above the level of the procedural call;
managers of expert system technology, who may end up being responsible for a program's decisions, need to satisfy themselves that a system's mode of reasoning is applicable to their domain.

The topic of explanations sometimes goes under the name of transparency, meaning the ease with which one can understand what the program is doing and why. It interacts with the issue of control, mentioned in the previous section, because the steps of reasoning exhibited by the program will depend upon how it goes about its search for a solution. How best to manage the interaction between explanation and control is still a research question, which we address in Chapter 16.

Explanation is closely linked to the topic of evaluation, since it is by scrutinizing the outputs of a system and examining a trace of its reasoning that we decide whether or not that system is getting the right answer for the right reasons. Unless a system has a good explanation facility, an expert will be unable to assess its performance or give advice as to how its performance could be improved. Evaluation is a difficult task that requires a certain level of commitment from both the expert and the computer scientist (see Chapters 3, 13 and 17).

**The Riddle of the Picture**

A well-known riddle has a man looking at a portrait and saying:

‘Brothers and sisters have I none, but this man’s father is my father’s son.’

Who is the man in the picture? First, take a moment to solve this puzzle. Second, imagine how you would explain the solution to someone who could not work it out, without using any props, such as pencil and paper. Many people find this puzzle hard to solve, and may even find it hard to follow the solution (Smullyan, 1978).

The answer is that the man in the picture is the son of the man looking at the picture. If we use a logical representation, this can become clearer. Let Pete be the man in the picture and Luke be the man looking at the picture.

\[ \text{son(father(luke), father(pete))} \]

‘this man’s father is my father’s son’

\[ \text{for all } X, \text{ if son(father(luke), X) then X = luke.} \]

Thus son is a relation between two people, but father can be represented by a function, since everyone has a single father. Stated in this way, it becomes obvious that

\[ \text{father(pete) = luke.} \]

by substitution, so Luke is looking at his son’s picture.

The right representation can sometimes make problems easier to solve and solutions easier to understand. But there is still an art to choosing such representations and presenting such solutions. And explanations do not always take the form of a proof, as we shall see in Chapter 16.

---

**1.4 Summary and chapter plan**

In this section, I try to be frank about the limitations of expert systems, as well as emphasizing their strengths, so that the general reader has at least some idea of how applicable this technology might be to his or her area. Then I give an overview of the rest of the book, together with some hints about which chapters to read, depending upon your interests. Such information is also provided in the preface, but since no one (including myself) ever reads prefaces, I reproduce it here.

**1.4.1 What is the state of the art?**

The main question that occurs to a potential consumer of expert systems technology is ‘Will it solve my problem?’ and the answer is ‘That depends.’ The three most critical factors are the nature of the task, the availability of a certain kind of expertise, and the ability to analyze that task and that expertise in such a way that a computer program, using rather limited forms of reasoning, can work out what has to be done. The following points cover most of the ground.

Someone (your expert) must:

- be able to perform the task;
- know how they perform the task;
- be able to explain how they perform the task;
- have the time to explain how they perform the task;
- be motivated to cooperate in the enterprise.

These conditions will tend to rule out certain applications from the start. Weather prediction, for example, is not a task that anyone performs very well – not even the professionals. Speech recognition is a task that we all perform extremely well, but none of us (including linguists) have much idea how we do it, so knowledge-based methods have so far met with less success than statistical modeling. Even given a genuine expert with insight into his or her skills, your application depends crucially upon that person’s ability and willingness to explain their skills in detail. These conditions may not be met. Your expert may be too inarticulate or too busy to take part in the knowledge engineering enterprise. Experts are usually in demand, and they often prefer doing what they do best to talking about it. They may be jealous of their expertise, perhaps fearing that its mechanization will threaten their livelihoods. There may be lack of commitment on the part of management to make adequate resources available to the engineers.

Even if the above conditions are met, there may be features of the task that limit the extent that the skills can be mechanized, for example:

- if the task involves complex sensory-motor skills beyond the scope of current technology in robotics and computer vision;
- if the task involves commonsense reasoning or arbitrary amounts of everyday knowledge (see Chapter 3).

It is useful to contrast the kind of knowledge required to become an expert in some field with the kind of knowledge that one needs just to get to and from work. Navigating busy streets in a vehicle requires rather more in the way of scene analysis and
hand-eye coordination than the present generation of robots. But one wouldn’t want to
call human drivers ‘experts’ (certainly not in my home town of Rochester, New
York).

It is also useful to consider the enormous amount of knowledge that we all possess
about the world: knowledge of objects and their properties, people and their motiva-
tions, physical causality and likely courses of events – the list appears to be endless.
This collection of hazy perceptions, vague intuitions and general principles certainly
isn’t expertise, but nonetheless we still have only the most rudimentary notions about
how to impart this kind of information to computers. So any task that is not sufficient-
ly self-contained to be encapsulated in a finite set of facts and rules is definitely beyond
the start of the art.

On the other hand, problems which can be solved by the enumeration of
associations between observable patterns of data and classes of events are well suited
to this technology. For example, operational problems in engineering systems, such
as heating, ventilation and air conditioning, can be monitored and diagnosed by
rule-based systems which correlate energy consumption signatures of buildings and
environmental parameters with underlying causes. Also, problems which involve con-
structing some object, by either selecting and fleshing out a skeletal plan, or combining
primitive components, are within the state of the art. Thus, in Chapters 14 and 15, we
encounter systems which configure elevator and computer systems.

1.4.2 Chapter plan of the book

Table 1.1 summarizes the topics we discussed in Section 1.3, and provides pointers to
those chapters which have most to say about them. The general reader might wish to
concentrate on knowledge representation and control of reasoning, since these are the
core topics of expert systems. Knowledge acquisition and explanation of solutions are
crucial topics also, but they relate more to the practical side of getting information
into and out of a real system that will see operational use.

Chapters 2 and 3 introduce some basic concepts in expert systems. Chapter 2 con-
tains a brief survey of those developments in artificial intelligence which created the
intellectual climate in which expert systems research was conceived and conducted.
Chapter 3 is also introductory in nature, in that it describes an early expert system and
explains why it was built and how it works.

Chapters 3–9 cover the main representation schemes for encoding domain-specific
knowledge in programs, in such a way that the knowledge can be applied to complex
problems by a computer. We begin with a brief overview of symbolic computation and
then proceed to explore a number of special-purpose representation languages, such as
CLIPS. We also consider the use of more general-purpose object-oriented languag-
es, such as C++, in the construction of expert systems. Finally, we address the problem
of approximate reasoning, and introduce various quantitative and qualitative meth-
ods for coping with uncertainty.

Chapters 10–16 deal with the practical, engineering side of expert systems technol-
yogy. We begin with the problem of knowledge acquisition – that is, how to elicit
knowledge from a human expert and codify it before representing it using the tech-
iques described in earlier chapters. Subsequent chapters consider a number of prob-
lem solving paradigms which have been found suitable for tasks such as diagnosis and
design, and illustrate them with exemplars from the literature. These exemplars were
chosen for pedagogical reasons, rather than because they were necessarily the ‘best’ in
the field. Nevertheless, they do include some success stories of expert systems research,
and there are lessons to be learned from the way in which these systems were designed
and implemented.

Chapters 17–19 examine software tools and architectures. We begin by taking a
critical look at the kinds of programming tool and programming environment typical-
ly provided for building expert systems. Then we describe two additional frameworks
around which expert systems can be organized: blackboard systems and truth main-
tenance systems.

In the remaining chapters, we touch on some more advanced topics, such as ma-
chine learning, belief networks, case-based reasoning, and hybrid systems. The final
chapter contains a summary of the book, recommends some topics for further study,
and discusses a few outstanding problems.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Definition</th>
<th>See Chapters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acquiring knowledge</td>
<td>Transfer of expertise from a person to a program</td>
<td>10–15, 20</td>
</tr>
<tr>
<td>Representing knowledge</td>
<td>Encoding expertise in a machine usable form</td>
<td>2–9, 21–23</td>
</tr>
<tr>
<td>Controlling reasoning</td>
<td>Deciding when to apply what piece of knowledge</td>
<td>3, 11–12, 17–19</td>
</tr>
<tr>
<td>Explaining solutions</td>
<td>Transfer of results from program to person</td>
<td>3, 8, 16, 22</td>
</tr>
</tbody>
</table>

Table 1.1 Summary of expert system topics and guide to chapters of this book