A hair perhaps divides the false from true.
—The Rubaiyat of Omar Khayam, translated by Edward Fitzgerald.

1 Introduction

The objective of this book is simple: to enable the reader to construct successful real-world fuzzy expert systems for problem solving. Accomplishing this objective, however, is not simple. We must not only transmit a good deal of knowledge in several different fields, but must also transmit the skills necessary to use this diverse knowledge fruitfully. To do this, we have concentrated on techniques that are simple in themselves, and that embody considerable power in problem solving. The examples we have chosen to illustrate these techniques are also must often quite simple. But do not be misled; there is a vast difference between “simple” and “trivial”.

Our scope will be, of necessity, somewhat larger than the preceding paragraph might indicate. In real life, we solve problems by thinking about them; we must therefore deal with the emulation of human thought by a computer program. In real life, we do not often think about problems as conventional computers do; we deal constantly with uncertainties, ambiguities, and contradictions. We sometimes use deductive logic, but more often we think intuitively, assembling information relevant to a problem, scanning it and coming to a conclusion. Besides this, we can often learn from our experience.

Expert systems tend to be viewed as retarded children by conventional artificial intelligence practitioners. Indeed, a conventional expert system may not have sufficient capabilities for meaningful emulation of thought. There may be two reasons for this: insufficient capability for emulating complex thought patterns; and lack of capability for understanding natural language. FLOPS addresses the first of these problems; it does not address the second, and still relies on a formal language. The FLOPS language does, however, permit emulating thought patterns of considerable complexity, including two different types of learning.

This book will have far fewer references than would normally be expected in a book of this type. There is a reason for this. Fuzzy systems theory has had a major impact on the field of process control, and in this field there are many references to be found. But this book reports theory and methods for general purpose reasoning, a much more difficult task than process control, and a field in which there has been very little research and very few meaningful papers published. There are books by Kandel (1991) and Kasabov (1998) that deal with fuzzy expert systems, but not in the detail necessary to actually construct one other than for control, nor in the requirements that the emulation of though places on general-purpose fuzzy reasoning systems.

Computer programming of rule-based fuzzy expert systems can be difficult for those trained in Western dichotomous thinking. There are three novel concepts involved: *fuzzy systems theory*; *nonprocedural data-driven programming*; *and parallel programming*. Fuzzy systems theory permits handling uncertainties, ambiguities, and contradictions. Nonprocedural data-driven programming means that when FLOPS is in run mode and firing (executing) rules, the order in which rules are fired has nothing to do with the order in which they appear in the program, but only on the available data and on which rules or blocks of rules are enabled or disabled for firing. (Conventional procedural languages execute their statements *sequentially* in the order in which they are written, except for explicit transfers of control.) *Parallel programming* language means that all fireable rules are executed effectively at once instead of some predefined order.

In Chapters 1 and 2, we treat the basic programming problems. Chapters 3–5 deal with the requisite fuzzy mathematics involved; Chapters 6 and 7 treat methods of fuzzy reasoning, that is, inferring new truths. Chapter 8 deals with fuzzy expert system shells, the integrated development environments for constructing fuzzy expert systems. Chapters 9–12 handle increasingly sophisticated problem-solving techniques. Finally, Chapter 13 considers real-time on-line expert systems in which the data are automatically acquired from an external source. Because of the number of concepts and techniques that are unfamiliar to many readers, we have occasionally covered a topic more than once in different contexts to avoid flipping back and forth in the book.

1.1 CHARACTERISTICS OF EXPERT SYSTEMS

Expert systems are computer programs, designed to make available some of the skills of an expert to nonexperts. Since such programs attempt to emulate the thinking patterns of an expert, it is natural that the first work was done in Artificial Intelligence (AI) circles. Among the first expert systems were the 1965 Dendral programs (Feigenbaum and Buchanan, 1993), which determined molecular structure from mass spectrometer data; R1 (McDermott, 1980) used to configure computer systems; and MYCIN (Shortliffe, 1976) for medical diagnosis. Since the mid-1960s there have been many, many expert systems created for fields ranging from space shuttle operations through hospital intensive-care-unit patient monitoring to financial decision making. To create expert systems, it is usual to supply a development environment and possibly a run-time module; these are called expert system shells, of which a number are available. We can view human knowledge as *declarative* (facts we have in stored in memory), and *procedural*, skills in utilizing declarative knowledge to some purpose.

Most AI practitioners do not consider expert systems as deserving the name of Artificial Intelligence. For example, Schank (1984, p. 34) states: “Real intelligence demands the ability to learn, to reason from experience, to *shoot from the hip*, to use general knowledge, to make inferences using gut-level intuition. Expert systems can do none of these. They do not improve as a result of experience. They just move on to the next if/then rule". The Random House unabridged dictionary gives one definition of learning as "the act or process of acquiring knowledge or skill"; certainly an expert system, by adding to its database of facts, acquires knowledge, and by adding new rules acquires skill. *FLOPS permits both of the learning techniques*. While we agree with Schank's categorization of "Real intelligence", we cannot agree that "Expert systems can do none of these". FLOPS can construct new rules and add to its existing database of factual knowledge, and in parallel mode can scan a database quickly without the inordinate systems overhead required by sequential systems. These capabilities deny Schank's blanket statement that expert systems are incapable of any of the characteristics of real intelligence.

There is a variety of ways in which the problem of creating computer programs to act like an expert has been approached (Jackson, 1999). The earliest employs *rule-based systems*, which use If-Then rules to represent the expert's reasoning process (*if* the data meet certain specified conditions, *then* take appropriate actions). There is a respectable body of opinion among cognitive scientists that a very significant part of human reasoning can be expressed by rules (Anderson, 1993); this view lends additional interest to rule-based systems. Other approaches include *semantic or associative nets* (Quillian, 1968), *frames* (Minsky, 1975) and *neural nets*, currently very popular in a wide variety of fields. Of these, clearly dominant are the complementary rule-based systems and neural net approaches.

We are concerned in this book with representing procedural knowledge by rules; ***IF*** the available facts meet certain criteria ***THEN*** do whatever the rule specifies. Declarative (factual) knowledge may be represented by stored data.

Whatever type of expert system is employed, we must consider what the prerequisites are for constructing a successful system. The two primary sources of knowledge are the skills of an expert in the field, and available historical data. Rule-based expert systems rely considerably on incorporating the skills of an expert in the problem domain, but relatively little on historical data; neural networks rely heavily on an extensive historical database, and relatively little on a domain expert.

1.1.1 Production Systems

We distinguish between two types of expert systems; *procedural* systems, written in conventional procedural languages such as C++, and *production systems*, that employ rules of the type "IF (the data meet certain specified conditions) THEN (perform the specified actions)". The "IF" part of the rule is called the *antecedent*; the "THEN" part is the *consequent*.

The "Tower of Hanoi" is a typical AI toy problem. We have three vertical spindles; on one spindle we have a number of disks of decreasing diameter from the bottom up. The problem is to move the disks from one spindle to another, one disk at a time, never placing a disk on top of one of smaller diameter. A production rule from this problem might be

```
rule (goal Fires if only one disk to move)
IF (in Spindles n = 1 AND source = <S> AND destination = <D>)
THEN
  write "*** move <S> to <D> ***\n",
  delete 1;
```

In the antecedent of this rule, Spindles is a data structure containing items `n` (number of disks on the spindle), `source` (name of the spindle from which we are moving a disk) and `destination` (name of the spindle to which we are moving a disk). If there exists an instance of Spindles in which `n` is 1, `source` has some value <S> (whatever it is) and `destination` has some value <D>, then we write a message to the screen and delete the instance of Spindles. A production system may contain dozens, hundreds, or even thousands of rules.

1.1.2 Data-Driven Systems

Of especial interest are *data-driven production systems*. Such languages are quite different from the common procedural languages like C, Pascal, Fortran, or Basic. In a procedural language, the statements that comprise a major part of the language are executed in the order in which they appear (unless a specific transfer of control is encountered); in a data-driven language, the rules that comprise a major part of the language are candidates for execution whenever the data satisfy the data specifications in the "IF" part of the rule.

If the data satisfy more than one rule at once, common rule-based languages such as the well-known OPS languages fire their rules *sequentially*, one at a time. First, we determine which rules are made fireable by the data. Next, a *rule-conflict algorithm* decides which of these should be executed (*fired*). The fireable rules that were *not* picked for firing are usually placed on a stack for firing later on in case no rules are newly fireable (backtracking). The selected rule is fired, that is the THEN part of the rule is executed, and we go back to looking for newly fireable rules.

1.1.3 Special Features of Fuzzy Systems

Most fuzzy expert systems provide for *parallel* firing of concurrently fireable rules; that is, all fireable rules are fired effectively at one time, emulating a parallel computer. Parallel operation has several advantages for fuzzy systems. A parallel language is especially appropriate when working with fuzzy sets: It makes programming easier and runs considerably faster than an equivalent sequential system. But sequential programming has advantages too in some cases; it is appropriate for eliciting information from a user when each question to be asked depends on the answer to the previous question. Accordingly, a fuzzy expert system language should provide both sequential and parallel rule-firing mode. These features of a fuzzy expert system language, unfamiliar to most programmers, creates a steep learning curve. The budding fuzzy expert system builder has to learn:

- Working with IF-THEN rules as the primary element of the language.
- Learning the basics of fuzzy systems theory: fuzzy sets, fuzzy logic, and fuzzy numbers.
- Learning a data-driven non-procedural language.
- Working with both sequential and parallel language execution.

To ease the entry into such novel languages, it is extremely important that the integrated program development environment (IDE) provide a solid base of help files to the user, as well as a variety of both simple tutorial and simplified real-world programs.

1.1.4 Expert Systems for Fuzzy Control and for Fuzzy Reasoning

There are two general types of fuzzy expert systems: fuzzy control and fuzzy reasoning. Although both make use of fuzzy sets, they differ qualitatively in methodology.

Fuzzy process control was first successfully achieved by Mamdani (1976) with a fuzzy system for controlling a cement plant. Since then, fuzzy control has been widely accepted, first in Japan and then throughout the world. A basic simple fuzzy control system is simply characterized. It accepts numbers as input, then translates the input numbers into linguistic terms such as Slow, Medium, and Fast (*fuzzification*). Rules then map the input linguistic terms onto similar linguistic terms describing the output. Finally, the output linguistic terms are translated into an output number (*defuzzification*). The syntax of the rules is convenient for control purposes, but much too restrictive for fuzzy reasoning; defuzzification and defuzzification are automatic and inescapable. There are several development environments available for constructing fuzzy control systems. A typical fuzzy control rule might be

- IF input1 is High AND input2 is Low THEN output is Zero.

Rules for fuzzy reasoning cannot be described so compactly. The application domain of fuzzy control systems is well defined; they work very satisfactorily with input and output restricted to numbers. But the domain of fuzzy reasoning systems is not well defined; by definition, fuzzy reasoning systems attempt to emulate human thought, with no *a priori* restrictions on that thought. Fuzzy control systems deal with numbers; fuzzy reasoning systems can deal with both numeric and non-numeric data. Inputs might be temperature and pulse, where temperature might 38.5°C and pulse might be 110 and "thready", where "thready" is clearly non-numeric. Output might be "CBC" and "Admit" and "transfer MICU" (not very realistic, but illustrates non-numeric data input and output). Accordingly, rules for fuzzy reasoning do not make fuzzification and defuzzification automatic and inescapable; they may be broken out as separate operations that may or may not be performed as the problem requires.

The syntax of fuzzy reasoning rules accepts a wide variety of rule types. Here are two.

1. IF symptom is Depressive and duration is ∼> about 6) THEN diagnosis is Major_depression;

   This rule resembles a fuzzy control rule, but it is actually quite different. In a fuzzy control rule, "symptom" would be a scalar number; in the fuzzy reasoning rule, "symptom is a (fuzzy) set of linguistic terms of which "depressive" is a member. Similarly, in a fuzzy control rule "diagnosis" would be a scalar number; in the fuzzy reasoning rules, "diagnosis" is a (fuzzy) set of diagnoses of which "depressive" is a member.
2. Rule block 8 (goal Generates rule for response to conditioned stimuli)

```
IF (in Count id1 = <ID2> AND id2 = <ID1> AND N > 2)
   (in Wired-in id = <ID2> AND response = <R>)
THEN
   rule block 3 (goal Conditions stimulus <ID1> to
   stimulus <ID2>)
   IF (in NewStimulus id = "<ID2>")
   THEN
     message '<ID1> - LOOK OUT - <ID2> coming\, <R>!\n';
```

A rule for learning, this rule has no counterpart at all in fuzzy control. Under specified circumstances, a new rule is created associating previously unassociated stimulus so that (e.g.) the burnt child learns to dread the fire.

1.2 NEURAL NETS

Neural nets (Haykin, 1994) do not require that the thinking patterns of an expert be explicitly specified. Instead, two sets of data are required from the real world. These data include all the inputs to the system, and the correct outputs corresponding to these input values. The first data set or *training set* is used to train the neural network so that, as nearly as possible, the correct outputs are produced for each set of input values. The second data set or *validation set* is used after the neural net has been trained to make sure that correct answers are produced on different input data. An advantage of neural nets is that it is not necessary to extract the thinking patterns of an expert and to render these explicit. Disadvantages are that a substantial training set is required, and that while the neural net may produce reasonably correct answers, in general we have little or no idea how it does this. Considerable work has been done on extracting rules from a trained neural net, but this work is not yet advanced to a very satisfactory state. A rough comparison between neural nets and expert systems is shown in Table 1.1.

TABLE 1.1 Comparison of Fuzzy Rule-Based Systems and Neural Nets

Property	Fuzzy Expert System	Neural Net
Data required to construct system	Minimal	Considerable
Expert knowledge required to construct system	Considerable	Minimal

Conditions under which neural nets may be the best approach include the following:

- There are ample data to form a training set of actual inputs and correct outputs corresponding to the inputs.
- No one has a good idea how outputs are related to inputs.
- We are not particularly interested in how inputs and outputs are related, so long as the system works.

The first of these conditions must be met; the second militates strongly in favor of a neural net; and the third helps incline us toward a neural net.

Conditions under which a fuzzy expert system is likely to be better than a neural net:

- We have a domain expert who knows fairly well how inputs and outputs are related.
- We do not have sufficient data to form a training set, possibly because the possible combinations of inputs and outputs are too numerous, or because collecting a training set would be prohibitively expensive.
- We are quite interested in the way in which outputs can be derived from inputs.

The first condition is almost essential, although a very skilled knowledge engineer may be able to become a domain expert with the dedicated help of persons less than fully expert in the understanding the task to be performed.

1.3 SYMBOLIC REASONING

Key to expert systems (and to AI generally, for that matter) is the concept of *reasoning with symbols*. (Ordinary procedural computer languages such as C and Fortran use symbols, but in a more restricted context.) In procedural languages such as C, symbols can represent numerical or character string data, or a collection of such data in data structures. Symbols can also represent logical propositions (simple Boolean comparisons between simple data items), program flow control by such constructs as "if . . .", "for . . .", and "while . . .", and smaller subprograms (functions). In object-oriented programs, symbols can represent *objects*, collections of both data items and functions. However, for the *symbolic reasoning* required by AI, symbols can represent almost anything, and languages more appropriate than

C or FORTRAN for symbol manipulation are used, such as LISP and PROLOG. For expert systems, specialized AI-based languages for rule-based reasoning such as the OPS family, Prolog, and CLIPS can be used.

For fuzzy expert systems, using familiar words as symbols to represent such concepts as fuzzy sets, fuzzy numbers, uncertainties, and modifying words (adjectives and adverbs) called *hedges*, special languages are available such as METUS (Cox, 1999); FLOPS, a fuzzy superset of OPS5, described in this book; and FRIL, a fuzzy superset of PROLOG (Baldwin et al., 1995). There are also special languages for fuzzy process control, but while very good for that purpose they lack the generality usually expected of an expert system language.

The buzz-word phrase "Computing with Words" has been very popular with fuzzy systems people for some years now. However, their use of words has been largely confined to words that describe numbers. This severe and very unfortunate limitation has greatly handicapped developing the enormous potential power of fuzzy systems theory. FLOPS is intended to permit emulation of human thought, although it has a long way to go to complete its objective; consequently, we must be concerned with words in a much more general sense than fuzzy control programs. At a minimum, we must think of words as parts of speech: nouns (e.g., age), simple verbs (equals), adverbs (approximately), conjunctions ("and") and adjectives (about), and define these appropriately (in the semantic sense) and precisely (say in Backus-Nauer Form) for our computer language.

1.4 DEVELOPING A RULE-BASED EXPERT SYSTEM

Rule-based systems require that the expert's knowledge and thinking patterns be explicitly specified. Usually, two persons (or groups) develop a system together. These are the *domain expert*, who knows how to solve the problem at hand but who is seldom acquainted with computer programming; and the *knowledge engineer*, who is thoroughly familiar with the computer technology involved and expert systems, but who usually has little or no knowledge of the problem at hand. Obtaining this knowledge and writing proper rules is called the *knowledge acquisition* phase (Scott et al., 1991). After the system has been written, it must be tuned for accuracy using a tuning data set similar to the training set of a neural net, but usually much smaller. After tuning, a rule-based system must be validated in the same way as a neural net. Rule-based systems have two advantages. A large training set is usually not required, and since the expert's thinking is explicitly spelled out, we now know how he thinks about the problem. They have the disadvantage that the knowledge acquisition phase may be difficult. A great advantage of fuzzy expert systems is that the rules can be written in language that the expert can directly understand, such as "if age is about 40" or "if patient is very old", rather than in computer jargon. Communication between domain expert and knowledge engineer is thus greatly eased.

Another advantage of rule-based expert systems is their ability to learn by creation of new rules. Probably the first example of a rule-based expert system to rival human experts was DENDRAL, which deduced the molecular structure of organic compounds from knowledge about fragments into which the compound had been broken (Jackson, 1999, pp. 383 ff). One set of DENDRAL's programs worked directly with the data to produce candidate structures. An additional program, Meta-DENDRAL, worked directly with the DENDRAL rules to improve them and discover new rules, thus discovering new concepts about the data. Meta-DENDRAL was not itself written as a rule-based expert system, but the ability of a rule to generate new rules and new expert factual knowledge opens the possibility for writing an expert system that can create new rules and store new expert factual knowledge. This exciting possibility has not as yet been well explored, perhaps due to the general (and, we think, incorrect) assumption that expert systems are no longer to be considered as artificial intelligence.

Many years ago the British scientist Alan Turing proposed a test for machine intelligence. In one room, we have a computer terminal with a human operator. This master terminal is linked to two other rooms. In one of these rooms we have a computer; in the other, a terminal with a human operator. If the operator at the master terminal cannot detect which of the other two rooms has the computer, then the computer's program can be called intelligent. There is no restriction on how the computer program does what it does; its performance is all the Turing test considers.

Our interest is not in passing the Turing test. Instead, we are interested in the nature of a computer program; we would like it to be constructed so as to emulate the thought processes of a human and particularly those thought patterns that can be verbally expressed. If a computer program *acts* as if it were intelligent, but does so through symbol manipulations that have little or no connection to normal thought, we are not especially interested.

As we have mentioned, a substantial body of thought, stemming largely from the work of Artificial Intelligence pioneer Allen Newell, contends that much verbal reasoning can be successfully expressed in *production rules*, called here simply *rules* (Anderson, 1993). We subscribe to this line of thought. Rules take the form "IF the data available meet certain specified conditions THEN take these specified actions", in which "actions" should be viewed in a very broad context, including drawing conclusions, firm or tentative. A sample simple rule might be "IF the car engine will not turn over when attempting to start THEN check if the battery is discharged". A more complex fuzzy rule might be "IF the pulmonary artery systolic pressure is considerably reduced AND the diastolic pressure is at least normal THEN the pressure reading might be damped".

1.5 FUZZY RULE-BASED SYSTEMS

Fuzzy rule-based systems, in addition to providing convenient handling of uncertainties of values (which can be done in other ways), furnish several additional capabilities. Approximate numerical values can be specified as fuzzy numbers. Numerical input values can be easily translated into descriptive words such as

"Fast" or "Large". Ambiguities and contradictions are easily handled by discrete fuzzy sets. Modifying words such as "very" and "slightly" are easily incorporated into the rule syntax. The approximate versions of the full range of Boolean numerical comparisons such as "$\sim<=$" or "approximately less than or equal to" are easily implemented. Most of our experience with fuzzy rule-based systems has been gained over the past 15 years using the fuzzy expert system shell FLOPS (Siler et al., 1987), and this book will rely heavily on that language. We have shared experiences and ideas with Earl Cox (1999, 2000), who developed the fuzzy expert system shell Metus, and who has deployed many expert systems using that language. We are also aware of the important PROLOG-based shell FRIL by Baldwin et al. (1995). The FLOPS language is still under development, but it does provide most of the features discussed here.

Fuzzy rule-based systems capable of both sequential and parallel rule processing add additional features to conventional expert system languages: convenient handling of uncertainties, ambiguities and contradictions, and modifying words such as "very" and "somewhat". Thus fuzzy systems increase our ability to emulate the non-rigid thinking patterns of (say) physicians and biologists as well as the relatively rigid patterns of (say) computer scientists. It is very unfortunate that most fuzzy expert systems have been devoted to control applications, and hence have concentrated almost exclusively on symbols that represent numerical quantities. While the enormous success of fuzzy control systems makes this understandable, it means that the ability of fuzzy rule-based systems to emulate reasoning with both numeric and non-numeric quantities remains to a large extent unexplored. We hope that this book will help illustrate the capabilities and potentialities of fuzzy rule-based systems for the emulation of thought in a much more general sense.

Most if not all human thinking is initially nonverbal. However, we have become reasonably successful in translating some thought into words (How can I say this?); knowledge engineers spend a great deal of time with domain experts converting their thought patterns into words, an hence to rules. Some nonverbal thought can be expressed in mathematical terms (e.g., feature extraction from images). A decent expert system language should enable writing a computer program using both words and mathematical transformations.

1.6 PROBLEMS IN LEARNING HOW TO CONSTRUCT FUZZY EXPERT SYSTEMS

The first problem for the knowledge engineer is to become adept at a different kind of computer language. Next is the problem of acquiring relevant knowledge from the domain expert. Finally, there are the problems of writing, debugging, calibrating, and validating the expert system itself.

Rule-based expert systems, and especially fuzzy expert systems, pose some learning problems for the programmer used to procedural languages such as C, Fortran, or Basic. Rules are basically IF statements: IF (this is true) THEN (execute the following instructions.) The IF part of a rule is called the *antecedent*, and the

THEN part is called the consequent. At this level, there does not seem to be any substantial difference between a rule-based expert system and a conventional C or Fortran program. However, the most powerful expert systems are not executed in a step-by-step procedural fashion, in which statements are executed in the order in which they appear, unless a specific transfer-of-control instruction is encountered. Instead, many expert system programs are *data-driven*; that is, rule are executed (fired) whenever the data sufficiently satisfy the antecedent, regardless of the sequence in which the rules appear in the program. This immediately creates the first problem—what do we do if the data satisfy the antecedents of two or more rules simultaneously? If we select one of these rules for firing next, what do we do with the fireable but unfired rules? Alternatively, we might elect to fire all the concurrently fireable rules effectively in parallel. Very few programmers have any experience with parallel languages—most of us have experience only with languages in which the instructions are executed sequentially. Another problem involves running time—if we change a piece of data, must we then examine all the rules over again to see which ones are now newly fireable? Charles Forgy's RETE algorithm goes a long way to solving that problem, but running time for a data-driven expert system is still long compared to procedural language programs, a price paid for the flexibility of data-driven systems and their inherent similarity to human thought patterns.

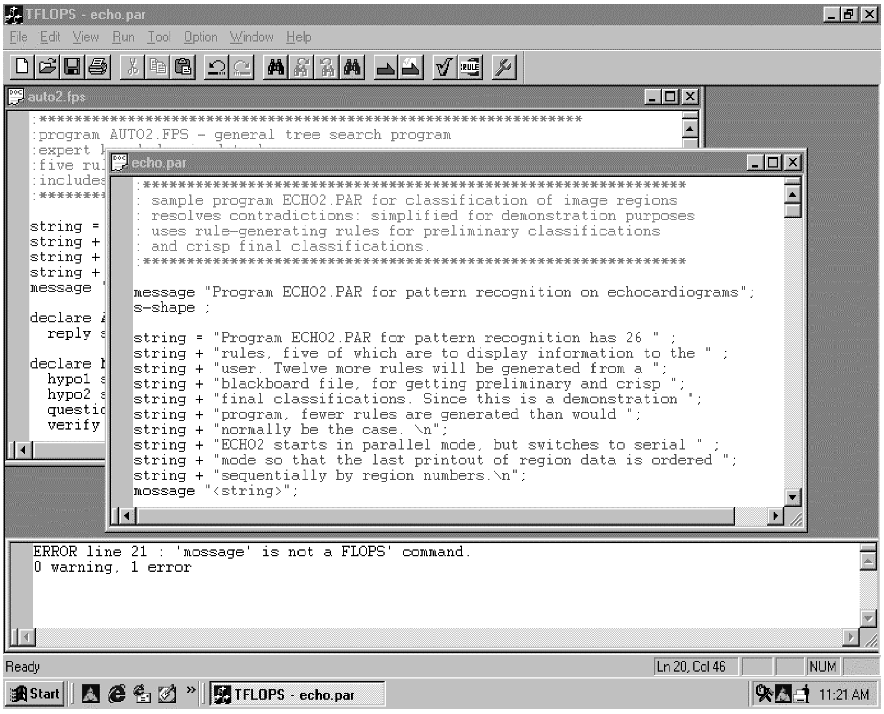
Another learning problem involves the use of multivalued logic. In a fuzzy expert system, things may be completely true, completely false, or anything in between. Ambiguities abound in fuzzy systems, and contradictions are frequently encountered; fuzzy systems provide structured ways of handling uncertainties, ambiguities, and contradictions, none of which are ordinarily encountered in conventional computer programming. Fuzzy systems also employ some special terms and concepts: fuzzy sets, fuzzy numbers, and membership functions. Monotonic and non-monotonic reasoning become routine concepts and acquire special meaning.

All this means that a fair amount of effort is required to become fluent in a fuzzy expert system language. There are, however, some bright points. You will be able to write rules using common words such as "very slow", "somewhat", and "roughly 60", making communication between knowledge engineer and domain expert much easier.

1.7 TOOLS FOR LEARNING HOW TO CONSTRUCT FUZZY EXPERT SYSTEMS

While this book aims to provide the reader with some theoretical and practical knowledge about fuzzy expert systems (which might be useful in passing an examination), we also aim to teach the reader a *skill* in constructing these systems in the real world. It is impossible to teach a skill without practice; if the reader does not actually run and write programs, he will no more learn this skill than a person who tries to learn to play tennis by reading books and listening to classroom lectures.

The CD Rom that accompanies this book has important tools to aid the reader. First, there are the two major programs: FLOPS itself, the inference engine and run-time



```
:****************************************************************
:program AUTO2.FPS - general tree search program
```

```
:*************************************************************
: sample program ECHO2.PAR for classification of image regions
: resolves contradictions: simplified for demonstration purposes
: uses rule-generating rules for preliminary classifications
: and crisp final classifications.
:*************************************************************

message "Program ECHO2.PAR for pattern recognition on echocardiograms";
s-shape ;

string = "Program ECHO2.PAR for pattern recognition has 26 " ;
string + "rules, five of which are to display information to the " ;
string + "user. Twelve more rules will be generated from a ";
string + "blackboard file, for getting preliminary and crisp ";
string + "final classifications. Since this is a demonstration ";
string + "program, fewer rules are generated than would ";
string + "normally be the case. \n";
string + "ECHO2 starts in parallel mode, but switches to serial " ;
string + "mode so that the last printout of region data is ordered ";
string + "sequentially by region numbers.\n";
mossage "<string>";
```

```
ERROR line 21 : 'mossage' is not a FLOPS' command.
0 warning, 1 error
```

Figure 1.1 Typical TFLOPS screen, with two FLOPS programs being edited and an. . . error log from a FLOPS run.

environment for executing and debugging FLOPS programs; and TFLOPS, the IDE for writing and testing FLOPS programs. Extensive help files are available in both FLOPS and TFLOPS, including an on-line manual and basic fuzzy math tutorial. Second, there is a fairly large number of example FLOPS programs, both tutorial and simplified versions of working expert systems. Third, there are simple specialized programs to illustrate some theoretical points made in this book. The reader is urged to make full use of these tools, especially TFLOPS for editing and running FLOPS programs. A typical TFLOPS screen is shown in Figure 1.1.

1.8 AUXILIARY READING

While this book is designed to be self-sufficient for our major purpose, we also recommend some auxiliary reading for readers with special interests. Jackson (1999) gives excellent coverage of non-fuzzy expert systems, although his treatment of fuzzy expert systems leaves very much to be desired. Klir and Yuan (1996) give in-depth coverage of modern fuzzy mathematics. Scott et al. (1991) cover issues of knowledge acquisition by the knowledge engineer. Anderson (1993) discusses rules from the viewpoint of a cognitive scientist interested in how the mind functions.

Finally, Cox (1999) presents much information on fuzzy expert systems from the viewpoint of finance and management; his discussion of modifying words used in rules, such as “somewhat”, “very”, and so on (called *hedges*) is unsurpassed, and his experience in constructing real-world fuzzy expert systems is extensive.

1.9 SUMMARY

Expert systems are computer programs designed to bring an expert’s skill to solving a particular problem. The most common types of expert systems are rule-based programs and neural networks. These differ considerably in the availability of expert knowledge and data required to construct the system, although there is little difference in the data required to validate the system once constructed.

Production systems are rule-based systems whose rule syntax is “IF (the data satisfy these specified conditions) THEN (perform these specified actions”. The “IF” part of the rule is the antecedent; the “THEN” part is the consequent. Production systems are most often data driven; that is, the eligibility of a rule for firing depends solely on the data, and is independent of the rule’s placement in the program. During a program run, the truth value of a rule’s antecedent is calculated; if the antecedent is sufficiently true, the rule is eligible for firing. If sequential rule-firing has been chosen, one of the rules whose antecedents are sufficiently true is chosen for firing by a rule-conflict algorithm; if parallel rule-firing has been chosen, all fireable rules are fired, effectively in parallel. Because of the features of a fuzzy expert system language which are unfamiliar to most programmers, it is especially important to have a good IDE, which makes available good help files on a number of topics. Special debugging tools should be available.

Two types of persons usually develop rule-based expert systems: domain expert and knowledge engineer. Ideally, the domain expert is thoroughly familiar with the problem domain and how to solve problems that arise in that domain, but who may be completely ignorant of computer programming and expert systems. The knowledge engineer, in contrast, is thoroughly familiar with expert system tools and with the construction of expert systems, but may be completely ignorant of the problem domain. These two work together on the expert system’s construction, sharing knowledge and skills.

1.10 QUESTIONS

1.1 In what three respects does FLOPS differ from conventional programming languages?

1.2 How does a data-driven non-procedural language differ from conventional procedural languages?

1.3 What can a language using fuzzy mathematics do that conventional programming languages cannot?

1.4 What is the difference between a parallel and a sequential program?

1.5 In what major respect can FLOPS programs conform to Roger Schank's definition of intelligence?

1.6 In what fundamental respects do expert systems differ from neural networks?

1.7 How do expert systems for fuzzy control differ from those for fuzzy reasoning?

1.8 What two kinds of people are usually involved in the construction of an expert system?

1.9 In what major respects do fuzzy expert systems differ from nonfuzzy systems?

1.10 What problems are encountered when first constructing a fuzzy expert system?

1.11 What important tool should be available for constructing a fuzzy expert system?