2
The architecture of expert systems

Richard Forsyth

The most significant thing about expert systems is that they are highly successful: already there are systems that can out-perform skilled humans at medical diagnosis, mass-spectrogram interpretation predicting crop disease, prospecting for minerals and much else besides. Suddenly people and indeed large corporations are using AI programs to get rich. Expert systems have finally disposed of the old dictum: if it works, it's not AI!

An expert system is based on an extensive body of knowledge about a specific problem area. Characteristically this knowledge is organized as a collection of rules which allow the system to draw conclusions from given data or premises.

This knowledge-based approach to systems design represents an evolutionary change with revolutionary consequences; for it replaces the software tradition of

Data + Algorithm = Program

with a new architecture centred around a 'knowledge base' and an 'inference engine', so that now

Knowledge + Inference = System

which is clearly similar, but different enough to have profound consequences.

2.1 FEATURES OF EXPERT SYSTEMS

But what exactly is an expert system? The British Computer Society's Specialist Group on the subject has proposed a formal definition.

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

That says it all, but personally I prefer an informal definition: 'An expert system is a piece of software that causes TV producers to lose all sense of proportion.'

In fact, you may find a list of distinctive features more useful:

(1) An expert system is limited to a specific domain of expertise.
(2) It can reason with uncertain data.
(3) It can explain its train of reasoning in a comprehensible way.
(4) Facts and inference mechanism are clearly separated.
(Knowledge is NOT hard-coded into the deductive procedures.)
(5) It is designed to grow incrementally.
(6) It is typically rule-based.
(7) It delivers *advice* as its output -- not tables of figures, nor pretty video screens but sound advice.
(8) It makes money. (This is a performance requirement.)

Up to now the biggest problem has been getting the knowledge from an expert into a machine-manipulable form.

2.2 COMPONENTS OF AN EXPERT SYSTEM

We have said that an expert system contains an inference engine and a knowledge base. There are in fact four essential components of a fully fledged expert system:

(1) The knowledge base;
(2) The inference engine;
(3) The knowledge-acquisition module;
(4) The explanatory interface.

All four modules shown in Fig. 2.1 are critical, and while a knowledge-based system may lack one or two of them, a truly 'expert' system should not. Let us look at each of them in turn.

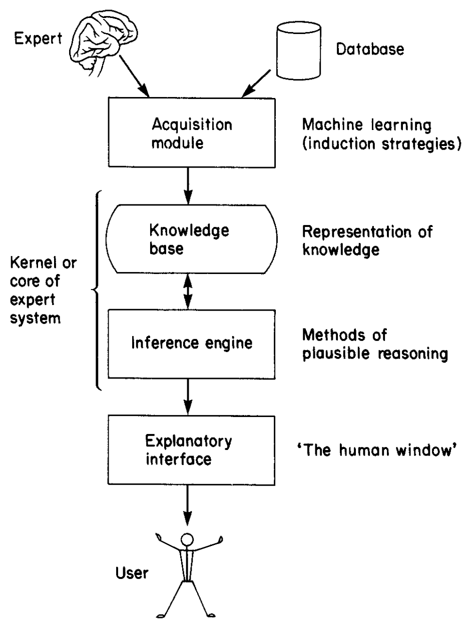
Expert
Database
Acquisition module
Machine learning (induction strategies)
Knowledge base
Representation of knowledge
Kernel or core of expert system
Inference engine
Methods of plausible reasoning
Explanatory interface
'The human window'
User

Fig. 2.1 A typical expert system

[Fig. 2.1 A typical expert system.]

2.3 THE KNOWLEDGE BASE

A knowledge base contains facts (or assertions) and rules. Facts are short-term information that can change rapidly, e.g. during the course of a consultation. Rules are the longer-term information about how to generate new facts or hypotheses from what is presently known.

How does this approach differ from conventional database methodology? The major difference is that a knowledge base is more creative. Facts in a database are normally passive: they are either there or not there. A knowledge base, on the other hand, actively tries to fill in the missing information.

Production rules are a favourite means of encapsulating rule-of-thumb knowledge. These have a familiar IF-THEN format, for example:

```
Rule 99
IF      the home team lost their last home game,
        AND the away team won their last home game
THEN    the likelihood of a draw is multiplied by 1.075;
        the likelihood of an away win is multiplied by 0.96.
```

But remember, these rules are not embedded in program code: they are data for a high-level interpreter, namely the inference engine.

Production rules are not the only way to represent knowledge. Other systems have used decision trees (e.g. ACLS), semantic nets (e.g. PROSPECTOR) and predicate calculus. Since one form of predicate calculus -- 'Horn clauses' -- is built into PROLOG together with a free theorem prover, this latter representation shows signs of gaining in popularity. At some very deep level all kinds of knowledge representation must be equivalent, but they are not all equally convenient. When in doubt, the best plan is to choose the simplest you can get away with.

2.4 THE INFERENCE ENGINE

There is some controversy in the field between supporters of 'forward chaining' versus 'backward chaining' as overall inference strategies. Broadly speaking, forward chaining involves reasoning from data to hypotheses, while backward chaining attempts to find data to prove, or disprove, a hypothesis. Pure forward chaining leads to unfocused questioning in a dialog-mode system, whereas pure backward chaining tends to be rather relentless in its goal-directed questioning.

Therefore most successful systems use a mixture of both, and Chris Naylor (1983) has recently described a method known as the Rule Value approach which combines some of the merits of both strategies. I call it 'sideways chaining'. (See also Chapter 6 of this volume.)

Whether your inferencing procedure works primarily backwards or forwards, it will have to deal with uncertain data and this is where things start to get interesting. For too long computer specialists have tried to force the soft edges of the world we actually inhabit (and understand well enough) into the rigid confines of hard-edged computer storage. It has never been a comfortable fit. Now we have means of dealing with uncertainty, in other words with the real world rather than some idealized abstraction that our data-system forces us to believe in.

Indeed, we have too many ways of dealing with uncertainty! There is fuzzy logic, Bayesian logic, multi-valued logic and certainty factors, to name only four. All sorts of schemes have been tried, and the odd thing is that most of them seem to work.

My explanation for this state of affairs is that the organization of knowledge matters more than the numeric values attached to it. Most knowledge bases incorporate redundancy to allow the expert system to reach correct conclusions by several different routes. The numbers measuring degree of belief are primarily for fine-tuning.

Hence, within reason, you are free to pick the measure of certainty that suits you best. (See also Chapters 5 and 6 in this volume.)

Let us move on to the question of knowledge acquisition.

2.5 INDUCTION STRATEGIES

Knowledge is a scarce and costly resource. How do we get hold of it?

Up to now the main bottleneck in the development of expert systems has been acquiring the knowledge in computer-usable form. Experts are notorious for not being able to explain how they make decisions, and the explanations they do give often turn out to be mere window-dressing. How then can we formalize their expertise?

The traditional way has been to closet a highly paid 'domain expert' (or two) with a highly paid 'knowledge engineer' for a period of months, during which time they effectively negotiate a codified version of what the expert knows. This process takes time and, obviously, money.

So there is a keenly felt need to automate the knowledge acquisition process. In my opinion Doug Lenat's EURISKO program (1982) is the forerunner of a new generation of machine-learning systems.

Lenat is not the only worker active in this field. Michalski (1980) has designed a system which learned to classify soya-bean diseases. Quinlan (1979) devised a concept-learning algorithm that grows its own decision trees by looking at a database of examples. I myself wrote a program (Forsyth, 1981) that uses a Darwinian scheme, termed 'naturalistic selection', to create and amend classification rules. It is called BEAGLE (Biological Evolutionary Algorithm Generating Logical Expressions).

But the striking thing about EURISKO is that its description language (the notation in which it stores rules and concepts) is expressive enough to allow it a rudimentary self-consciousness, in the form of 'meta-rules'. It is rather an introspective system, and spends a lot of time monitoring its own performance, recording its findings as rules that apply to itself.

One of its discoveries had an ironic twist. It noticed that human rules tended to be better than its own, so it came up with the heuristic:

```
IF      a rule is machine-made,
THEN    delete it.
```

Luckily the first machine-generated rule that EURISKO erased was that one!

(Knowledge acquisition and refining is dealt with more fully Chapters 10, 11 and 12 of this volume.)

2.6 THE HUMAN WINDOW

The fourth main component of a true expert system is the explanatory interface.

One of the best things about the classic expert systems like MYCIN is the care exercised over the user interface. At any time the enquirer can ask the system why it made a given deduction or asked a particular question. Rule-based systems generally reply by retracing the reasoning steps that led to the question or conclusion. The ease of doing this is a point in favour of rule-based programming.

The explanation facility should not be regarded as an optional extra. Donald Michie (1982) and others have warned about the dire consequences of systems which do not operate within the 'human cognitive window', i.e. whose actions are opaque and inexplicable.

If we are to avoid a succession of Three-Mile-Island-type disasters or worse, then our expert systems must be open to interrogation and inspection. In short, a reasoning method that cannot be explained to a person is unsatisfactory, even if it performs better than a human expert.

2.7 WHO NEEDS EXPERT SYSTEMS?

At this point the reader may be asking: Do I need an expert system? The answer depends on the kind of problem you want to solve. Table 2.1 is a checklist of features that affect the suitability of the knowledge-based approach.

Table 2.1

Suitable vs.	**Unsuitable**
Diagnostic	Calculative
No established theory	Magic formula exists
Human expertise scarce	Human experts are two-a-penny
Data is 'noisy'	Facts are known precisely

If your intended application falls more on the left than on the right, you should seriously consider an expert system.

By a diagnostic problem we do not mean just medical diagnosis, but include any area where there are several possible answers and the difficulty is selecting the right one, or at least filtering out the improbable ones. Many categorization and prediction tasks fall into this framework, e.g. computer fault diagnosis. Once you know that it is the disc interface rather than the main memory that is playing up, the hard part is (or should be) over: the repair is a matter of routine board swapping.

By a domain where no established theory exists we mean topics like tax law, motor repair, weather forecasting and indeed most branches of medicine. There are too many variables for a complete and consistent theory, so skilled practitioners rely on knowledge and 'intuition'. We exclude, by contrast, problems where you can plug in a formula, crank a handle, and out comes the answer -- for example, planetary motion, where Newton's laws are good enough for spacecraft guidance.

An area where human experts are scarce is easily recognized by the telltale symptoms of high salaries, job hopping and queues for vocational training courses. It is clearly cost-effective to try to computerize skills that are in demand (such as oil-well log interpretation) before more commonplace ones like image recognition which even pigeons can do, and which often turn out to be much harder to computerize than so-called intellectual skills.

Finally, if your information is reliable and clear-cut, an expert system is not particularly recommended. If on the other hand your data is 'dirty', it may be just what you need. Then fuzzy, fizzy, fudgy or some other funny logic can really come into its own.

2.8 LANGUAGE ISSUES

There is a widespread misconception that all expert systems have to be written in LISP or PROLOG.

LISP has proved its worth over 25 years of AI research but it is no accident that it has not caught on commercially. It thrives in academic hotbeds like MIT, Stanford and Carnegie-Mellon University, because it provides a medium for the interchange of ideas. You can take Joe Hacker's pattern-matcher, add Mike McQwerty's theorem-prover and bolt on young Carl Whizzkid's natural language parser. By that time you're more than half way there (as long as they speak to each other).

But once outside the AI sub-culture, you are on your own. LISP itself is hardly more than CAR, CDR, COND and CONS (plus EVAL). All the interesting bits are accretions which follow the LISP philosophy but which are in essence additional software packages. They are not standardized, and in the UK most of them are not even available. So just getting LISP really does not get you very far.

As for PROLOG, my advice is to forget it. PROLOG is a much-touted logic programming language heralded by some as the dawn of a new era in software development. If the Japanese do adopt it without radical modification (and the signs are that they will not), the West can relax till the end of the century. Perhaps the kindest thing to say about PROLOG is that it is ahead of its time. Now let's say some unkind things about it.

(1) PROLOG has very little error-protection against minor misspellings and so forth, which merely lead to bizarre unintended effects.
(2) The user must understand the implementation details of the backtracking mechanism to write code -- i.e. it is not true 'logic programming'.
(3) The order of clauses is crucial to their meaning -- again, this is contrary to the spirit of logic programming.
(4) Many built-in predicates have side-effects, rendering PROLOG unsuitable for efficient parallel execution.
(5) PROLOG provides a relational database for free -- a big bonus, but the trouble is that it resides in main memory, and is consequent very greedy on storage.
(6) Everything in the database is global: there are no local facts or modules in the accepted sense.
(7) Depth-first search is done for you -- whether you want it or not. If you want something more sophisticated, you have to 'cut' your way out of the jungle by hacking branches off the search tree.
(8) PROLOG is already riddled with non standard 'enhancements'.

In short, PROLOG is a hacker's delight, since only a dedicated elite can master its elegant intricacies. The claim that it is a logic programming language simply will not bear serious scrutiny. Some of my best friends use PROLOG, but I wouldn't touch it with a keyboard. When you hear of schoolchildren using it fluently you can be sure that they are being taught a subset which leaves out the 'knobbly bits' (and therefore would not suffice for serious software development).

As Feigenbaum and McCorduck (1983) put it: 'The last thing a knowledge engineer wants is to abdicate control to an "automatic" theorem-proving process that conducts massive searches without step-by-step control exerted by knowledge in the knowledge base.'

If you must use PROLOG, wait until it has been digested by BASIC. BASIC is like a ravenous python, devouring all that lies in its path. It has just finished swallowing PASCAL with all its control structures. After short pause and a few burps it will be ready to gobble up PROLOG, and we shall see BASICs that offer built-in resolution theorem-proving. Then it will be time to consider switching to PROLOG. (But see also Chapter 7, of this volume.)

2.9 DO IT YOURSELF

The best plan is to use the language you know on the machine you have. One relatively cheap solution is to purchase an expert system shell (e.g. Micro Expert or Nexus), use it as a prototyping tool till you know what you want, then write the production version in an efficient and portable language like C, FORTRAN 77, PASCAL or even FORTH.

You don't have to wait for the promised fifth-generation dream machines either. As long as your computer is not bigger than a VAX nor smaller than a Sinclair QL it can do the job. An IBM PC or Apple Macintosh, especially with a hard disc, would do quite nicely.

2.10 CONCLUSION

When it comes to knowledge-based systems, ignorance is just not a viable option. Nobody in the information business can afford a wait-and-see attitude, because the future has already begun.

REFERENCES

Feigenbaum, E. and McCorduck, P. (1983) *The Fifth Generation*, Michael Joseph, London.
Forsyth, R. (1981) BEAGLE: a Darwinian approach to pattern recognition. *Kybernetes*, 10.
Lenat, D. (1982) The nature of heuristics. *Artificial Intelligence*, 19, 189-249.
Michalski, R.S. and Chilausky, R.L. (1980) Learning by being told and learning from examples. *International Journal of Policy Analysis and Information Systems*, 4.
Michie, D. (ed.) (1982) *Introductory Readings in Expert Systems*, Gordon and Breach, New York.
Naylor, C. (1983) *Build Your Own Expert System*, Sigma Technical Press/John Wiley, Chichester.
Quinlan, J.R. (1979) Induction over Large Databases, *Report HPP-79-14*, Stanford University, Palo Alto, California.