Introduction

A particularly challenging assignment for a computer scientist is the design of a course on the frontiers of his subject for students in an engineering discipline. Such students are accustomed to different kinds of material and different styles of presentation. They prefer their courses to be quantitative rather than descriptive, analytic rather than synthetic, specific rather than general, and factual rather than speculative. They are suspicious of discursive lectures that cover seemingly unconnected topics and strive to pull the threads together at the end. They are extremely intolerant of anything that smacks of soft-science "waffle".

An equally serious problem is the lack of understanding of the foundations of computer science. Engineering students generally regard the subject as synonomous with programming, frequently in archaic and unstructured languages, which have been taught with inadequate emphasis on modern principles of software design. They are ill-prepared to absorb sophisticated material at the cutting edge of software research.

And yet graduating engineers certainly need to know about expert systems. Not, perhaps, about the technology as such, for it is moving so rapidly that current techniques are certainly transitory. But the new emphasis on knowledge-based systems, and the new direction of advanced programming environments towards supporting multiple paradigms of programming, are important fundamental changes which will certainly have a dramatic effect on the practice of engineering in the future.

This paper describes a course on expert systems designed and given in the Department of Electrical and Electronic Engineering at the University of Canterbury, NZ while the author was a visitor there. It occupied 16 one-hour lectures during March and April, 1986. Although primarily designed for final-year undergraduates, graduate students were permitted to take it for credit too. Such was the level of interest in the subject that many research students, academic staff, and members of other Departments attended the course as well, almost doubling the number of registered students.

A key problem in designing the course was the need to strike a balance between a practical view of the subject and its deeper foundations in artificial intelligence and philosophy. It was felt desirable to appeal to engineers through a practical orientation; yet clearly the subject will travel far and fast in the next few years and a foundational approach gives the best chance to impart knowledge of lasting value. Above all, it was deemed essential to avoid an airy, hand-waving treatment and concentrate on material that engineers felt they could get their teeth into. The fact that any expert system worthy of the name is singularly complex — far too complex to understand fully within the confines of an academic course — means that the subject is generally approached in one of two ways. One takes the high, discursive, road; through guidelines, case histories, evolutionary considerations, and the like. The other takes the low, superficial, road, with wide-ranging lists of existing expert systems, capsule descriptions, lists of knowledge engineering tools, application areas, and so on. Although both these approaches have their uses, neither was deemed suitable for the course. In particular, existing textbooks were felt to be completely unsatisfactory for our purposes.

The paper is structured as follows. The next section gives the aim of the course as it is explained to students. Following that we review the material that is covered and the approach taken. Heavy use is made of PROLOG to exemplify the concepts taught, and the rationale for this is discussed next. Finally we describe the laboratory, which forms a crucial component of the course, and close with some general conclusions. An extensive (65 page) set of course notes was prepared, which includes substantial and detailed laboratory exercises and has now been augmented with the final examination, and model answers to exam questions and assignments (Witten, 1986). This is available on request.

Aim of the course

The course aims to acquaint students with the field of "expert systems", which has recently become a sort of shop window for techniques of artificial intelligence. The emphasis is practical rather than theoretical. In particular, on completing the course students should be able to

- assess whether a particular problem is suitable for tackling with an expert system
- identify the key elements which distinguish expert systems from "ordinary" computer programs
- explain how a computer program might "explain" its reasoning
- distinguish different strategies used by rule interpreters to infer conclusions from evidence
- say what is meant by "object-oriented programming" and show how it relates to the use of "frames" for representing knowledge
- discuss the representation of uncertainty, and distinguish fuzzy logic from probability logic
- write simple expert system shells in PROLOG.

These aims are, of course, somewhat ambitious for a two-month course with no relevant prerequisites! Neverless it was felt that most students achieved them — albeit in a minimal sense. Many of them were illustrated by "toy" programs in PROLOG that the students worked hard to build themselves.

It would have been attractive to present examples which tackled engineering problems that students are familiar with or could at least relate to — like circuit analysis, filter design, electronic trouble-shooting, integrated circuit layout, computer configuration, or image interpretation. But in the event it was considered quite impossible to present detailed descriptions of any real expert systems within the scope of the course. In several places where an example was required, the medical expert system MYCIN was used (Shortliffe, 1976; Clancey & Shortliffe, 1984; Cendrowska & Bramer, 1984). The unfamiliarity of the domain certainly created problems, and it would have been much better to use an engineering system instead, but the compromise was made because MYCIN is so well documented, and time pressed.

Course outline

The course divides into five parts of roughly equal prominence: introducing expert systems, logic programming in PROLOG, knowledge as rules, knowledge as frames, and dealing with uncertainty. It was originally hoped to review in addition the topics of explanation and natural language, and knowledge acquisition techniques; however, time did not permit. Here the material in each part is briefly described, except that issues surrounding PROLOG are deferred until the next section.

Introducing expert systems answers some of the immediate questions — What are expert systems? How did they arise? Why is explanation so important? What kinds of domain are suitable? What are the current limitations? What systems are used commercially? It aims to satisfy initial curiosity and so leave the ground clear for detailed study of the techniques used in expert systems. A toy "expert" system is presented in full detail by showing transcripts of consultations, the rules which underlie them, how the rules are interpreted to direct the question and answer session, and how explanations are generated. The system used is actually a simple version of Hammond's (1984) questionnaire interpreter.

Knowledge as rules covers the simplest and most popular technique for constructing expert systems (Hayes-Roth, 1985). It describes what rules are, how they are expressed, and how they can be executed by a forward- or backward-chaining rule interpreter. But all non-trivial expert systems need more than just rules: it is at least necessary to have data structures for the rules to work on. The classic MYCIN medical expert system is described as an example of the infrastructure required to support a backward-chaining consultation system. Finally, some problems are noted which plague rule-based knowledge representation schemes — unforeseen

interactions, shallow reasoning, and superficial (though seductive) explanation capabilities (eg Cendrowska & Bramer, 1984; Clancey, 1979).

Knowledge as frames begins with an introduction to the object-oriented programming paradigm, including an implementation in PROLOG of a rudimentary form of object-oriented programming (Zaniolo, 1984). This allows some of the issues surrounding the concept of "inheritance" of attributes in an object hierarchy to be raised in a concrete way (Brachman, 1983). Next, frame systems are introduced as more structured, more elaborate, schemes for representing objects and their attributes. As a detailed example, the UNIT frame system is presented (Stefik, 1979; Rawlings & Fox, 1983), together with its recent descendant KEE (Fikes & Kehler, 1985). In order to demonstrate that rather different kinds of frame systems exist, this part of the course closes with a brief description of the epistemologically-based KL-ONE frame package (Brachman & Schmolze, 1985).

Dealing with uncertainty introduces the manifold problems of representing uncertain knowledge in expert systems. Bayesian inference is the standard technique, and is covered first. In practice, however, one is virtually always forced to adopt some unjustified assumptions of independence of evidence. Opinions differ as to whether or not this is a real problem (eg Cheeseman, 1985, argues against the numerous critics of Bayesian techniques). The designers of MYCIN certainly thought it was, and proceeded to develop their own calculus of uncertainty, briefly described next, to overcome the shortcomings of the Bayesian model (Shortliffe, 1976). Unfortunately (for them) this approach presented serious problems (Adams, 1976). It is now known that significant advantage can be gained within classical probability logic by representing uncertainty by intervals rather than points (Quinlan, 1983; Nilsson, 1986). But a more general development is possible which subsumes both fuzzy logic and classical probability logic as special cases, and is covered in the course (Gaines, 1984).

Use of PROLOG

In both lectures and laboratory, the course places strong emphasis on programming — even though students lack any prior exposure to an AI programming language. It was decided early on that strategies of rule interpretation, property inheritance, automatic explanation, and so on should be exemplified through actual code fragments. After much discussion and reflection on LISP *vs* PROLOG, the latter was chosen for the following reasons — many of which are subjective and controversial.

- PROLOG is a smaller language and therefore less daunting to students.
- Toy expert systems can be built directly in it more easily than in raw LISP.
- Although the LISP-based OPS-5 package was available, PROLOG allows a greater variety of techniques to be exemplified.
- Good programming environments were available for neither; but this impacts PROLOG less seriously than LISP.
- Although PROLOG is an unusual language, our previous experience in introducing it to Engineering students at Calgary indicated that they may find it easier to grasp than do Computer Science students who are hampered by their wider experience of more conventional programming languages.

It is emphasized in the course that PROLOG had been selected on its merits as a pedagogical tool (possibly one which represented an investment for the future), and that existing industrial-strength expert systems are generally based on LISP. In fact, this causes students some disappointment.

A serious problem with the use of any AI language in an Engineering department is the lack of local knowledge. The student grapevine is unable to answer questions and resolve problems because a nucleus of expertise does not yet exist. It was realized in advance that, for the first invocation of the course, this bootstrapping problem would cause students great difficulty. Consequently a carefully-graded set of laboratory exercises was devised to get people off the ground as soon as possible.

The following strategy is taken to teach the PROLOG language. Students read Clocksin's (1984) introductory paper, and a similar development, though using different examples, is covered in lectures. This gives students two perspectives on the material. A detailed transcript of all Clocksin's examples was prepared, using the slightly different notation of UNSW PROLOG which is available on the local computer system. Students work their way through the transcript on the terminal (longish functors were provided in files to save typing). This seems to give them confidence in the language after playing with it for only 2–4 hours, although they still have great difficulty creating PROLOG programs of their own — which only practice can allay. "Impure" parts of the PROLOG language are avoided (eg tampering with the database; cuts), although students are warned of their existence.

The laboratory

An engineering-style laboratory was designed for the course, with small, carefully-explained exercises undertaken each week. This was partly dictated by resource limitations — a terminal room was set aside for two afternoons; students booked two one-hour slots each week. It also has the advantage of fitting in with engineering students' expectations of laboratory activity. But the primary benefit is that more material can be covered in the short time available by using well-structured, step-by-step experiments than with the more open-ended project-style assignments usually found in Computer Science courses.

The first two weeks' activity involves getting to know the computer system — including rudimentary file management and editing — and the UNSW PROLOG system, including the transcript of examples mentioned earlier. The next experiment introduces simple manipulation of rules expressed in PROLOG. Code is provided which tests whether a rule mentions a particular fact (in either its antecedent or consequent), and students have to experiment with it and make minor modifications. Simple recursion is used to break conjunctive antecedents down into single facts. By now students are expected to be able to write straightforward programs themselves.

A signal advantage of PROLOG is that it allows exercises which illustrate and amplify points made in the lectures to be tackled, starting from scratch, in a period of a few weeks. The remainder of the laboratory constitutes three programming assignments, which between them illustrate the major components of a simple expert system implementation language (or "shell"). Assignment 1 is to construct both forward and backward chaining rule interpreters for a small set of rules. Students are already familiar with representing rules through the preceding exercise. But the task of creating original code comes as quite a shock, and shakes students out of any complacency engendered by the step-by-step approach of the previous exercises. At this point many suffer a crisis of confidence, and realize that there is something "difficult" in all this programming stuff after all!

Assignment 2 provides just enough exposure to the object-oriented programming methodology to drive home some of the complexities of frame-based representations of knowledge. Students are given a small object package which implements inheritance of methods, and must use it to pretty-print the whole of an object hierarchy. Assignment 3 returns to a rule interpreter like one constructed earlier and adds to it an "explanation" facility through which the user can find out why questions were asked. This assignment proved too difficult for all but a few, but most students were by now fantastically well motivated and went through a tremendous learning experience in attempting it.

To make it feasible to complete each assignment in the time available, students are given test data and examples of what their programs should do. Solutions do not have to be robust or perform any checking of input data. In some cases fragments of code are provided as well to reduce the size of the task. Model answers are handed out and discussed in class shortly after assignments were due, so that students can correct misconceptions quickly and build on success.

Conclusions

The course demonstrates that it is possible for bright engineering students to develop a worthwhile knowledge of practical issues in applied artificial intelligence through a short, intensive, course — starting from ground level. PROLOG was invaluable as a pedagogical tool, both to explain concepts and to motivate practically-oriented students. It provides a sort of epistemological workshop where students can sharpen their understanding of general issues by tackling specific examples. The highly-structured engineering-style laboratory seems to be a good way of bringing students quickly to a point where they can benefit from this tool.

The final examination showed that most students gained a detailed understanding of such things as the operation of frame systems, rule interpreters, and reasoning under uncertainty. In addition, it was felt that students achieved the wider aims of the course as stated earlier. And they enjoyed it.

Acknowledgements

Special thanks are due to John Andreae, the Department of Electrical and Electronic Engineering at the University of Canterbury, and Werner Staringer of Vienna. This work was performed while on sabbatical from the University of Calgary.

References

Adams, J.B. (1976) "Probabilistic reasoning and certainty factors" *Mathematical Biosciences, 32,* 177-186.

Brachman, R.J. (1983) "What IS-A is and isn't: an analysis of taxonomic links in semantic networks" *IEEE Computer, 16* (10) 30-36, October.

Brachman, R.J. and Schmolze, J.G. (1985) "An overview of the KL-ONE knowledge representation scheme" *Cognitive Science, 9* (ii) 171-216.

Cendrowska, J. and Bramer, M.A. (1984) "A rational reconstruction of the MYCIN consultation system" *Int J Man-Machine Studies, 20,* 229-317.

Cheeseman, P. (1985) "In defense of probability" *Proc 10th International Joint Conference on Artificial Intelligence,* 1002-1009.

Clancey, W.J. (1979) "Tutoring rules for guiding a case method dialogue" *Int J Man-Machine Studies, 11,* 25-49.

Clancey, W.J. and Shortliffe, E.H. (Editors) (1984) *Readings in medical artificial intelligence.* Addison-Wesley, Reading, Massachusetts.

Clocksin, W.A. (1984) "Introduction to PROLOG" in *Artificial Intelligence: tools, techniques, and applications,* edited by T.O'Shea and M.Eisenstadt. Harper & Row, New York.

Fikes, R. and Kehler, T. (1985) "The role of frame-based representation in reasoning" *Communications of the Association for Computing Machinery, 28* (9) 904-920, September.

Gaines, B.R. (1984) "Fundamentals of decision: probabilistic, possibilistic and other forms of uncertainty in decision analysis" *Studies in the Management Sciences, 20,* 47-65.

Hammond, P. (1984) "Representation of DHSS regulations as a logic program" *Proc Expert Systems 83,* 225-235, Churchill College, Cambridge, England, December.

Hayes-Roth, F. (1985) "Rule-based systems" *Communications of the Association for Computing Machinery, 28* (9) 921-932, September.

Nilsson, N.J. (1986) "Probabilistic logic" *Artificial Intelligence, 28,* 71-87.

Quinlan, J.R. (1983) "INFERNO: a cautious approach to uncertain inference" *Computer Journal, 26* (3) 255-269.

Rawlings, C. and Fox, J. (1983) "The UNIT package — a critical appraisal of a frame-based knowledge representation system" *Proc Expert Systems 83,* 15-29, Churchill College, Cambridge, England, December.

Shortliffe, E.H. (1976) *Computer-based medical consultations: MYCIN.* Elsevier, New York.

Stefik, M. (1979) "An examination of a frame-structured representation system" *Proc 6th International Conference on Artificial Intelligence,* 265-270.

Witten, I.H. (1986) "Expert systems" *Man-Machine Studies, UC-DSE* (28) 5-65, University of Canterbury, Christchurch, New Zealand, May.

Zaniolo, C. (1984) "Object-oriented programming in PROLOG" *Proc International Symposium on Logic Programming,* 265-270, Atlantic City, New Jersey, February 6-9.