CURRENT DEVELOPMENTS IN ARTIFICIAL INTELLIGENCE AND EXPERT SYSTEMS

by Donald Michie

Abstract. The definition of an expert system as a knowledge-based source of advice and explanation pinpoints the critical problem which confronts the would-be builders of such systems. How is the required body of knowledge to be elicited from its human possessors in a form sufficiently complete for effective organization in computer memory? This article reviews recent advances in the art of automated knowledge-extraction from expert-supplied example decisions. Computer induction, as the new approach is called, promises both important parallels to the human capacity for concept formation and also commercial exploitability.

The professional activity of the knowledge engineer is to develop expert systems. An expert system is a machine system which embodies useful human knowledge in machine memory in such a way that it can give intelligent advice and also can offer explanations and justifications of its decisions on demand. That is the key clause in the customer's specification. A system which gives good decisions but cannot explain itself in terms to which the human expert can relate may be a software product of great value, but it belongs to some other category: operations research, decision support systems, automatic control, and so on. So the system must be capable not only of emulating the expert in the quality of decisions, but also in the ability to give reasons and justification. From a structural point of view an expert system is a knowledge-based inference engine (see fig. 1).

The inference engine is an interpreter for a high-level language in which the knowledge base is expressed. The knowledge base itself is an interconnected set of pattern-coded hypotheses, observations, and rules. So we find a sharp contrast with classical, sequentially driven

Donald Michie is Director of Research and Advanced Study, The Turing Institute, 36 North Hanover Street, Glasgow G1 2AD, Scotland and professor emeritus of machine intelligence, University of Edinburgh. He presented this paper at the annual conference ("From Artificial Intelligence to Human Consciousness") of the Science and Religion Forum, Canterbury, England, in April 1984. This article is reprinted here, by permission, from the *International Handbook of Information Technology and Automated Office Systems* (first edition, 1985).

[*Zygon*, vol. 20, no. 4 (December 1985).] ISSN 0044-5614

programs. The typical setup is a division between a body of situation-action rules and what is called the database, which maintains an updated map of the current state of the problem (see fig. 2).

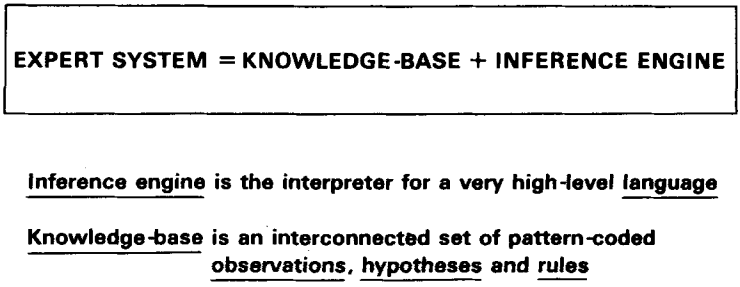

FIG. 1.—Structure of an expert system.

SITUATION: SOMETHING THAT MAY OR MAY NOT BE SATISFIED IN THE DATABASE

ACTION: SOME PROCESS THAT POSSIBLY CHANGES THE DATABASE

FIG. 2.—Partition of an expert system into "database" and "rulebase."

The driver of such a system is not the sequence in which the rules are written down, but the matching process whereby a rule is selected. The current state-description, or situation, is something that may or may not be satisfied in the database. That is to say the left-hand side, the "if" part of each given rule, may or may not match with something in the database. An action is some process: it may be simply drawing a conclusion, it may be putting out a message, it may be initiating a robotic action. But whatever it is, it is some process that possibly changes the database.

In the simplest case control simply cycles through the rules in the rulebase and in each cycle finds which rules have their situation part satisfied, uses some criterion of conflict resolution to select one of the candidate rules for selection and then performs the action part of that rule. The action may have the form of offering advice or may even be asking a question. If the user responds by querying the selected action, then the system will explain it, for example, somewhat trivially but

quite effectively by displaying the sequence of rules which generated its behavior. Figure 3 illustrates the recognize-act cycle.

EACH CYCLE THE INFERENCE ENGINE

- **FINDS WHICH RULES HAVE THEIR SITUATION PART SATISFIED IN THE 'DATABASE'**
- **SELECTS ONE OF THEM TO BE 'FIRED'**
- **PERFORMS THE ACTION PART OF THE SELECTED RULE (THUS POSSIBLY CHANGING THE DATABASE)**

THE "ACTION" MAY HAVE THE FORM OF OFFERING ADVICE OR ASKING A QUESTION

IF THE USER RESPONDS BY QUERYING THE ACTION THE SYSTEM MAY "EXPLAIN" IT BY DISPLAYING THE SEQUENCE OF RULES WHICH TRIGGERED IT.

FIG. 3.—The recognize-act cycle of an expert system.

The tradition of knowledge engineering as it has evolved in the United States has been based on a scenario in which the knowledge engineer, the computer scientist who specializes in doing this kind of work, labors hand in hand with a domain specialist whose knowledge it is desired to transfer. The engineer seeks to draw the required expertise out of the expert's head in the form of rules which can be encoded in the machine system. The so-called dialogue acquisition method is depicted in figure 4 as an old-style knowledge engineer's route map.

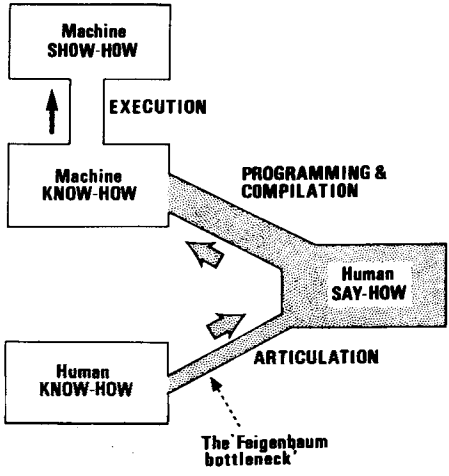

FIG. 4.—Knowledge engineer's route map: old style.

The knowledge engineer's object is to convert human know-how into "say-how" through a process of articulation of which the expert is supposed to be capable. Once in this form, the traditional arts of programming and compiling can convert it into machine code. This code is the machine representation of the know-how. At run time it generates what I have called machine show-how, in other words the expert behavior that the customer desires.

This seemed a promising way to go because experts in various domains, such as chemistry, troubleshooting complex equipment, or medical diagnosis, are surprisingly confident about their ability to access the large body of pattern based rules which they have in their heads. It seemed a reasonable idea to take this confidence at its face value. However, time has moved on, and this craft has been in existence for nearly ten years since the original laboratory demonstrations. Yet the number of systems which are out in the marketplace actually earning money could, I think, be counted on the fingers of two hands. There must be a reason for this.

Edward Feigenbaum has put his finger on the trouble. Note the channel in figure 4 which I have drawn as rather narrow. He called it "the bottleneck problem of applied artificial intelligence" (Feigenbaum 1977). If the narrowness of that channel is independent of the complexity of the task domain, then the situation is one which the technology could live with. We would have a hard time, and we would always have a hard time pushing or pulling expertise through the narrow channel, but it could at least be done. In the last few years at Edinburgh my laboratory thought it worthwhile to make an investigation to see if this hypothesis of independence is valid. There is an alternative possibility, namely, that the more complex the mental skill, the greater the proportion of it which is encoded in intuitive form and hence beyond access by anybody including the expert. The results of our quite extensive tests have been conclusive. Expert articulacy is *not* independent of complexity. One reaches the dark area of inaccessibility surprisingly soon as one moves up the complexity scale.

So two alternatives present themselves. The first is that the knowledge engineering enterprise seemed like a good idea at the time, but has now reached the end of its useful life. The alternative possibility is that this route map is incomplete and that there is some way of going from human know-how to machine know-how other than by the method of articulation.

The first indication that the latter might be the case comes from the elementary observation that when an expert is asked to perform a know-how transplant into a human apprentice rather than into machine memory, he does not in general proceed by articulating the

precepts and rules of his craft. Most of the work is done by presenting a cleverly graded and sequenced series of *tutorial examples*.

It thus appears there is another way of moving this conceptualized material into another agent provided that that agent is in a suitably prepared state. In the human case the agent is the apprentice. By "suitably prepared" we mean that the agent should be capable of learning by example.

This blocked channel can thus be circumvented if and only if a means can be found for moving the rules from the expert's head to machine memory via the language of examples rather than via the language of explicit articulation. For that we require effective algorithms for inductive inference to be executed by the recipient machine. Such algorithms must simulate the apprentice's ability to reconstruct from the tutorial examples a mental model of the master's know-how.

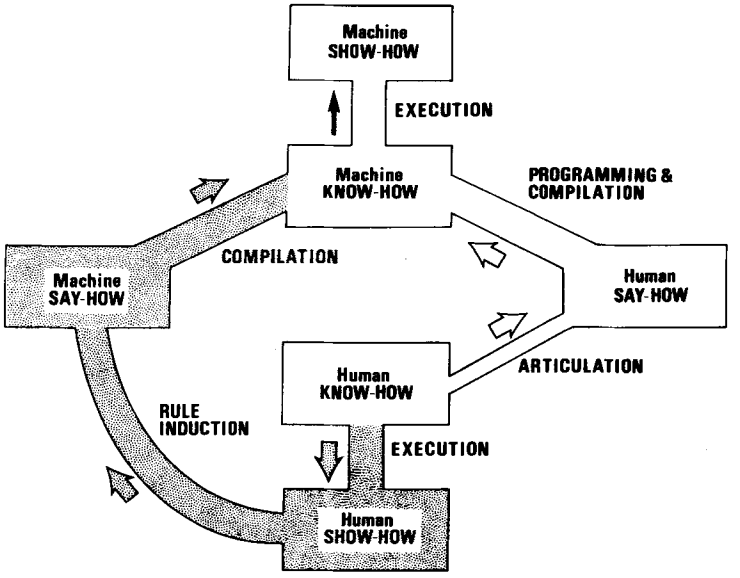

FIG. 5.—Map of overall problem.

Figure 5 depicts a map of the overall problem somewhat richer than the old-style map shown earlier. In this case we propose that it is possible to proceed from human show-how, that is, human-supplied examples. We know that the human can change his know-how into tutorial show-how. We know this because an expert is usually hired by the employer for two things, not just one. One task is to employ skills on actual cases: the second task is to transmit skills to new recruits. Given that, then by computer induction we can have the machine acquire

expert skills from tutorial examples of expert decisions presented to it. A model of the expert's skill in explicit form can then be displayed on the screen as rules. Those can be compiled into machine know-how. If this can all be done then we have a "northwest passage" to circumvent the bottleneck. Thanks to the fact that there are now some quite efficient inductive inference algorithms available it has been possible to engineer tool-kits for building expert systems from expert-supplied examples. In our own laboratory we do not now do things any other way.

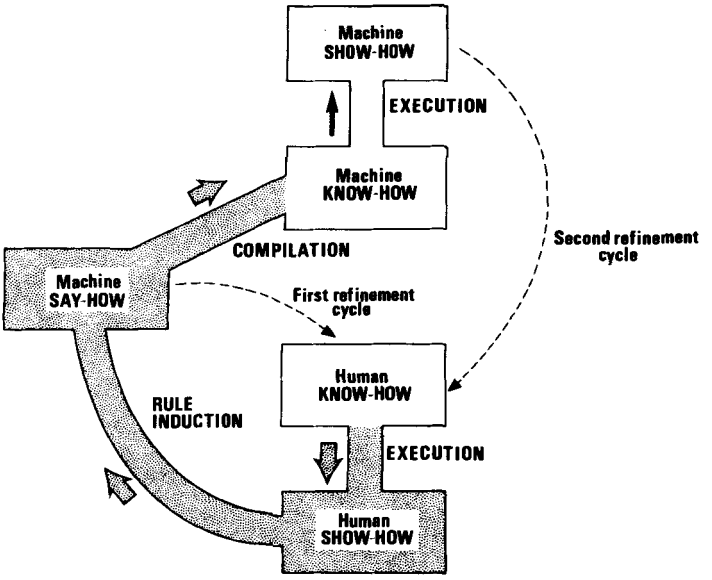

FIG. 6.—Knowledge engineer's route map: new style.

I do not want to suggest that we should make no use at all of expert articulation. There is no matter of principle about it. Rather, the question is where the main traffic should pass. What is involved is a cyclic trial and error process. The small refinement cycle in figure 6 depicts how the expert with the help of the computer scientist generates tutorial examples so that incrementally the machine can display successively refined rules on the screen; the expert then has the freedom to say whether she likes the rules. A vital criterion is, does she understand them? Do they make sense from her point of view? Second, can she mentally check them? If she can do both, she tends to be satisfied. If she is not satisfied, then she can use the rule editor directly and edit the rules. More often the domain specialist will go around the

cycle and generate new examples in order to refine or correct by induction those parts of the rulebase that she does not like. This design process is essentially iterative. Finally the rulebase is perceived as good enough to install in the field.

Inevitably there is a second refinement cycle occasioned by run-time "bugs" or errors, not low-level bugs of the kind with which programmers are mostly familiar, but more conceptual bugs. Customer complaints get reported back to the development laboratory. In due course the knowledge engineer has to sit down with his own domain specialist, or borrow one from the customer, and get these bugs out.

All this depends on there being good, economically adequate induction algorithms. There is quite a rich history of inductive inference work in artificial intelligence. One of the most active centers has for a long time been Ryszard Michalski's group at the University of Illinois. I shall not dwell on his work, interesting though it is. Whether or not the Michalski algorithms, which are academically motivated, could be made into cost-effective software tools is an open question. On the other hand, even earlier than Michalski's work, studies were reported in the 1960s which were overlooked by the artificial intelligence community and by the computer science community at large. This was work by Earl Hunt (1966). The algorithm is simple and was possibly neglected because Hunt is by profession a psychologist. So the work was published in the wrong subculture.

In 1978 I was teaching a graduate course in artificial intelligence at Stanford University. A distinguished visiting professor, Ross Quinlan, from Sydney decided to try the programming exercise which I had given to the class. The task was to use an induction algorithm to solve a certificated hard problem which I had brought from Edinburgh. The problem was hard in the sense that as far as we could tell, it was too complex to be programmable by conventional means, and therefore needed something special. As a doctoral student in the 1960s in Sydney, Quinlan had assisted Hunt. At the end of term, he had a program running in the programming language, Lisp, which solved the hard problem. Back in Sydney he recoded it in another programming language, Pascal, and proceeded to do a sequence of extremely interesting experiments (see Quinlan 1983).

Quinlan's own considerable extension of Hunt's algorithm is called ID3 and is the basis of all commercially viable induction systems at the present time. It is given a training set of positive and negative instances, where each instance is represented as a fixed list of attributes or properties. Attributes can have either truth values or numerical values. It is required to produce a decision tree for differentiating positive and negative instances. Since a decision tree is logically equivalent to a

conditional expression in a programming language, you can say that the output of the algorithm is a program. The synthesized expert program can then be run on new material, to test the level of skill induced by the particular examples used as the training set.

My Edinburgh laboratory subsequently developed an enhancement of ID3 to include the ability to handle numerical as well as logical attributes of the problem domain. A commercial version of this, Expert-Ease, is available on the IBM Personal Computer. Expert systems already developed with this approach include Dow-Jones forecasting, error-message interpretation in UCSD Pascal, classification of lymphatic cancers, developing rules for high school algebra and many others.

So far the source of the tutorial examples has been presented as being the human expert in interaction with the system, like a teacher with an apprentice. However, that was not the picture that Quinlan had in mind when he developed ID3. This point is brought out in figure 7. To apply ID3 or any other algorithm which generates rules from examples, that is, any algorithm for computer induction, a sufficient set of primitive attributes must have been identified. At present we do not know of any source other than the domain specialist that can say what primitives one ought to measure. However poorly developed the expert's powers of articulation of entire rules may be, she has no trouble in knowing what low-level measurements are relevant, although she commonly attributes relevance to additional measurements which later may be shown to be redundant.

ID3

Given: **a collection of positive and negative instances, where each instance is the description of an object in terms of a fixed set of attributes or properties**

Produces: **a decision tree for differentiating positive and negative instances**

FIG. 7.—The ID3 algorithm was conceived for operating on pre-existing collections of positive and negative instances of a concept.

This does not matter from the knowledge engineer's point of view. The induction algorithm simply leaves those attributes on one side and fails to incorporate them in the final rule. The expert may or may not then argue that it ought to have incorporated them. I am not implying

that the dummy attributes are not performing some important task in the cognitive economy of the expert. I suspect that they may be, just as various superstitious beliefs play a useful role in many highly developed skills including even in athletics. But logically from the point of view of incorporation into the final rulebase, they appear to be redundant.

A further requirement is an oracle, a source of expert decisions for the system to work from. The oracle can be one of a number of things. It can be a human oracle, that is, an expert, or it can be a precomputed database.

There is a well-known algorithm in two-person finite games with perfect information, which makes it possible to take some nontrivial endgame of only a few million legal positions in which chess masters flounder, and by exhaustive backwards computation develop a database, a lookup table, of positions paired with their game-theoretic values. By expending a great amount of computational labor, we end up with an oracle which is rather expensive to store. One application of computer induction is to take such giant lookup tables and squeeze them down into compact explanatory, descriptive, and predictive rules.

Another kind of oracle may be the external world. I will give two examples. One is the task of predicting what the Dow-Jones index is going to do in the next fourteen days. This was a student project at the University of Illinois and the student, Helmut Braun, selected it himself. He also located a professional group willing to act as certifiers of the quality of performance of the final system. In a graduate class we do not want expert systems which are only as good as human experts. That is for undergraduate students. In its run-time behavior it is reasonable to demand of properly engineered knowledge bases that they do a little better. The only way to find out in a given case is by getting authentic experts to spend some of their time validating student exercises. Braun found a company in Indianapolis that makes its money by publishing regular forecasts of the Dow-Jones index. Their success is based on the expert skill of the founder of the company.

There were thus two possible oracles that the student could use. He could use the skilled forecaster himself, since he had available an archival file of the last few years of his expert decisions. Or he could take what would in general be expected to give a higher livel of performance in the induced product, namely, what actually happened, not what the expert predicted. Did the index actually go up, or was it roughly stationary, or did it go down? Braun did it both ways and he got an "A" for that project. He also aroused interest in the accountancy department of the University, and the work is now being extended.

My other example of taking events in the real world as an oracle, that is, as a source of preclassified instances, is from the realm of control. We

tackled a task of this kind many years ago with extremely rudimentary induction methods compared with the present day. The task was that of balancing a pole on a motor driven cart that goes back and forth on a track. It must not drop the pole, and it also must not run off either end of the track—not a trivial task. An even more complex task would be riding a bicycle. If one received a contract to develop a robot bicyclist, the first task, as always, would be to work out a good set of measurements to take and a frequency of sampling. The human is not able to sample more than perhaps ten times per second. But the important thing is that the only sample of expert decisions is that obtained by monitoring the real-time behavior of a human bicyclist who is already able to perform this skill. This is what is meant by a real-world oracle.

Can an expert system be generated exclusively from examples? It has been done convincingly more than once, the first by R. Chilausky, B. Jacobsen, and Michalski in the middle 1970s in the case of diagnosing soybean diseases (see Michalski & Chilausky 1980a; 1980b). This is a crop of considerable commercial interest to the state of Illinois. They chose nineteen common diseases suggested by plant pathologist colleagues. These formed the attributes, some yes-no, some with result sets containing as many as seven different values (see fig. 8).

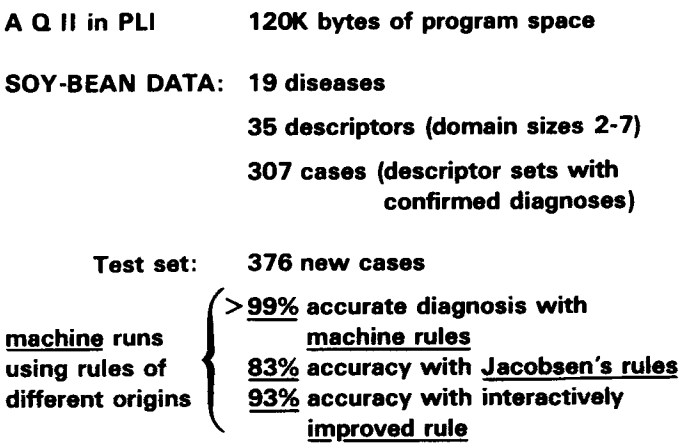

FIG. 8.—Expert system generated exclusively from examples compared with hand-crafted variants (from Chilausky, Jacobsen, and Michalski 1976).

Their training set comprised 307 cases, each presented as a list of 35 values paired with the name of the disease. They did the job both ways, that is, using dialogue acquisition and using inductive acquisition. They worked with Barry Jacobsen, a plant pathologist who specialized in soybean diseases and who was already well known for a taxonomy that he had developed himself. They got a rulebase by dialogue acquisi-

tion which, when tested on 376 new cases, gave 83 percent accuracy. This would have been encouraging were it not for the fact that by merely inducing rules from the data the machine obtained a rulebase of more than 99 percent accuracy. Only one plant out of the 376 was misclassified when the rule was tested on new material.

When Jacobsen saw that, he worked hard in dialogue acquisition mode using the computer facilities available, to see if he could improve his rules. He got it finally to 93 percent accuracy. He then decided to accept the machine-generated rules in place of his own as reference material, and this he uses to the present day. It is a significant early event—an example of a phenomenon that we will see more commonly. The useful products to be obtained from knowledge engineering have in the past been conceived entirely in terms of the run-time behavior of the program. But if the rulebase itself can be structured and constrained in such a way as to remain intelligible and professionally useable by the experts, then there is a product of a different kind, namely, new and improved codifications (see Shapiro & Michie [in press]). These have an independent value apart from whether they run on the machine. Such codifications are positive contributions in themselves and are indeed welcomed by the experts as new reference material.

Let us now consider Quinlan's induction system, ID3. We start off with a collection of instances which are already classified and then we proceed as in figure 9 (see Quinlan 1983). Having developed a rule, the algorithm uses it to classify new material sampled from the database. Each example for which the rule gives the right answer, the algorithm throws away. But whenever it finds a refutation example, it saves it, because that has "tutorial" value for creating a better rule. When the refutation file has grown to a preset limit then a new file of exceptions is merged with the old working set to form a new working set. The entire rule is then thrown away—rather shocking to many people because we humans do not work that way. We are conservative about our theories, and we try to hang on to them and patch them when we get refutations. But the economics of ID3 are such that it does better to forget the old rule, work instead on the augmented example set, and induce a new rule from scratch. In a really hard case it may go around the iteration loop 19 or 20 times, in the style shown in figure 9, until it has tested the current version of the rule on a sufficiently large number of new cases without finding a refutation—"sufficiently large" being a user-defined parameter. Then it terminates. Unless the resulting rule is tested on the entire universe of cases, there can be no guarantee that the rule obtained will be 100 percent correct. Quinlan's paper gives a technical discussion of that point, with some rather surprising and encouraging

results concerning bounded error (Quinlan 1983). The main result seems counter-intuitive: on the assumptions that Quinlan made in order to do the theory, accuracy of the rule is only a function of the size of the training set and is independent of the size of the universe. Second, one finds oneself also surprised at how accurate a rule can be obtained from how small a number of examples. Quinlan also reports systematic empirical tests. All of the results fall well within the bounds predicted by his theory.

ID3 was designed to work with large numbers of instances. It starts by selecting a small subset of them (the working set) and at each cycle

1. **Forms a decision tree that is correct for the current working set**
2. **Finds the exceptions to this decision tree among the other instances**
3. **Constructs a new working set from the current window and the exceptions**

FIG. 9.—Iterative refinement by ID3 of a classification rule in the form of a decision tree (see Quinlan 1983).

Quinlan also reports an extension of the original problem that I gave to the Stanford class. The original problem was from the chess end-game king-and-rook against king-and-knight. Is the knight side lost in two ply? In other words, is the position such that whatever the knight does, the rook side can either capture the knight or checkmate on the next move? This can be recognized more or less at a glance, certainly in a few seconds by a master. But if the object is to write a pattern-based program which can do fast "at-a-glance" recognition, it is an adverse programming problem. Quinlan then tackled the next level of complexity, knight lost in three ply. The rook side must spot whether there exists a move such that whatever the knight does, then the rook side kills him on the next move, either by capture or by checkmate. In terms of cognitive complexity, it is a substantial jump. Chess masters can take up to half a minute or so to classify such a position and in our experience just occasionally get it wrong. It is totally out of the question to build an expert system for this task in conventional style other than by reckless use of skilled manpower. Having elaborated a set of 49 primitive attributes, Quinlan generated a decision tree inductively using a training set of 715 instances. The decision tree had 177 nodes, that is,

84 tests in the corresponding conditional expression. It was fairly cheap to manufacture, requiring only 34 seconds on the Cyber 172 computer—leaning, it must be said, on much of the above-described human work as regards selection of primitive attributes. By contrast, the best expert system of the pattern-driven search variety that Quinlan could develop, using himself as both expert and knowledge engineer in internal dialogue, took him months of work. That is not a reasonable use of human resources to manufacture a program, when an equally accurate program can be manufactured in 34 seconds on a Cyber 172. But the denouement was even more striking. The run-time performance of the machine-made decision program was five times faster than the best specialized search program that Quinlan could code.

On the face of it, that looks like a very useful breakthrough for the software technology industry. A semiautomatic process can apparently generate programs which are actually five times more efficient than the best that programmers can code. There is no question that such programs and such ways of making programs will filter into software practice as soon as the industry becomes aware that these straightforward but powerful tools exist. However, there are grounds for unease. Quinlan's decision-tree program, although it ran fast and accurately on the machine, when shown to the chess master was completely opaque. It was not a question of a few glimmers of sense here and there scattered through a large obscure structure, but just a total blackout. We have repeatedly confirmed the phenomenon with other chess material in Edinburgh (Shapiro & Niblett 1982), and industrial associates have encountered it in, for example, circuit-board fault finding and process control. If this method of manufacturing useful bits of software has a very attractive feature, commercially attractive, does it matter if such artificially generated materials are opaque? In some applications it does not matter and a black box can be tolerated. But in the very complex decision-making areas where there is the greatest market pull, problems like control software for nuclear power stations, or, shall we say, the United States NORAD nuclear warning system, matters stand differently.

There are numerous areas where control software is already more opaque than is safe. We recently did a two-year study for the European Economic Community. What we found concerning opacity in hand-programmed systems was worrying enough. But a source of rather cheap automatically generated code which is intrinsically doomed to add to that opacity must be regarded as even more worrying.

For the user who needs transparency in the generated code, over very large and complex domains, a facility must be incorporated in the

induction package for the well-known strategy of "divide and conquer" common both to artificial intelligence and to the structured programming school. I am referring to the decomposition of a large problem into subproblems, sub-subproblems, and so on, thus forming a procedural hierarchy. This facility is not explicitly provided in Quinlan's algorithm. However, it is under development as a central feature of an inductively oriented expert systems language designed by Stephen Muggleton (1985) in Edinburgh. A version is available under the commercial name RuleMaster.[1] Preliminary results of its application to meteorological forecasting and to other problems are summarized below.

WILLARD is an expert system which forecasts the likelihood of severe thunderstorms occurring in the central United States. Examination and interpretation of critical meteorological parameters (such as temperature, moisture, winds, and pressure) by an experienced severe thunderstorm forecaster yields clues on the temporal and spatial domain of severe thunderstorms. The WILLARD expert system consists of a hierarchy of thirty modules, each of which contains a single decision rule. This hierarchy is on average four levels deep. All modules' rules were developed using inductive generalization (although RuleMaster allows expert authored rules). About 140 examples out of a possible nine million situations were used in building WILLARD. It can operate in interactive or batch forecast mode and gives a full explanation of reasoning on demand. WILLARD has been found to compare favorably with the United States National Severe Storm Forecast Center over a number of test cases.

EARL is an expert system for diagnosing faults in large distribution transformers. The repair or replacement costs of these large transformers range from $150,000 to $1,500,000. A human expert has in the past been employed in interpreting measurements of the amounts of different gases dissolved in the cooling fluid of these transformers; the relative quantities of these gases gives strong indications of possible problems in the transformer. This expert system has been installed and is being used by the Hartford Steam Boiler Insurance company. A recent comparison showed that the expert system made the same diagnosis as the expert it was based on in all 30 different test cases.

ARCH is a procedural expert system, built inductively, to carry out robot plans in a blocks world. The problem involved many of the facets of procedural expert systems, such as design and scheduling. ARCH, like the more diagnostic expert systems (WILLARD and EARL), gives, on demand, a full explanation of all decisions and actions made.

In conclusion, there is no compulsion for an information technologist to be interested in this new field, but when other methods fail there may be something of advantage here.

NOTE

1. RuleMaster is available through Intelligent Terminals Ltd. of Scotland and Radian Corporation, Austin, Texas. For an account of the design features, including back chaining and forward chaining and a finite state formalism conducive to control applications, see Michie, Muggleton, Riese, and Zubrick (1984).

REFERENCES

Chilausky, R., B. Jacobsen, and R. S. Michalski. 1976. "An Application of Variable-Valued Logic to Inductive Learning of Plant Disease Diagnostic Rules." *Proceedings of the Sixth Annual International Symposium on Multi-Variable Logic*, Utah.

Feigenbaum, E. A. 1977. *The Art of Artificial Intelligence 1: Themes and Case Studies of Knowledge Engineering*. Pub. no. STAN-CS-77-621. Stanford, Calif.: Stanford University, Dept. of Computer Science.

Hunt, E. B., J. Marin, and P. Stone. 1966. *Experiments in Induction*. New York: Academic Press.

Michalski, R. S. and R. L. Chilausky. 1980a. "Knowledge Acquisition by Encoding Expert Rules Versus Computer Induction from Examples: A Case Study Involving Soybean Pathology." *International Journal of Man-Machine Studies* 12:63-87.

———. 1980b. "Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis." *International Journal of Policy Analysis and Information Systems* 4:125-61.

Michie, D., S. Muggleton, C. Riese, and S. Zubrick. 1984. "RuleMaster: A Second-Generation Knowledge-Engineering Facility." In *First Conference on Artificial Intelligence Applications, Denver, 5-7 December 1984*, 591-97. Silver Spring, Md.: IEEE Computer Society.

Muggleton, S. 1985. "Inductive Acquisition of Expert Knowledge." Ph.D. diss., University of Edinburgh.

Quinlan, J. R. 1983. "Learning Efficient Classification Procedures and Their Application to Chess End Games." In *Machine Learning: An Artificial Intelligence Approach*, ed. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, 463-82. Palo Alto, Calif.: Tioga.

Shapiro, A. and D. Michie. In press. "A Self-Commenting Facility for Inductively Synthesised Endgame Expertise." In *Advances in Computer Chess, 4*, ed. D. Beal. Oxford: Pergamon.

Shapiro, A. and T. Niblett. 1982. "Automatic Induction of Classification Rules for a Chess Endgame." In *Advances in Computer Chess, 3*, ed. M. R. B. Clarke. Oxford: Pergamon.