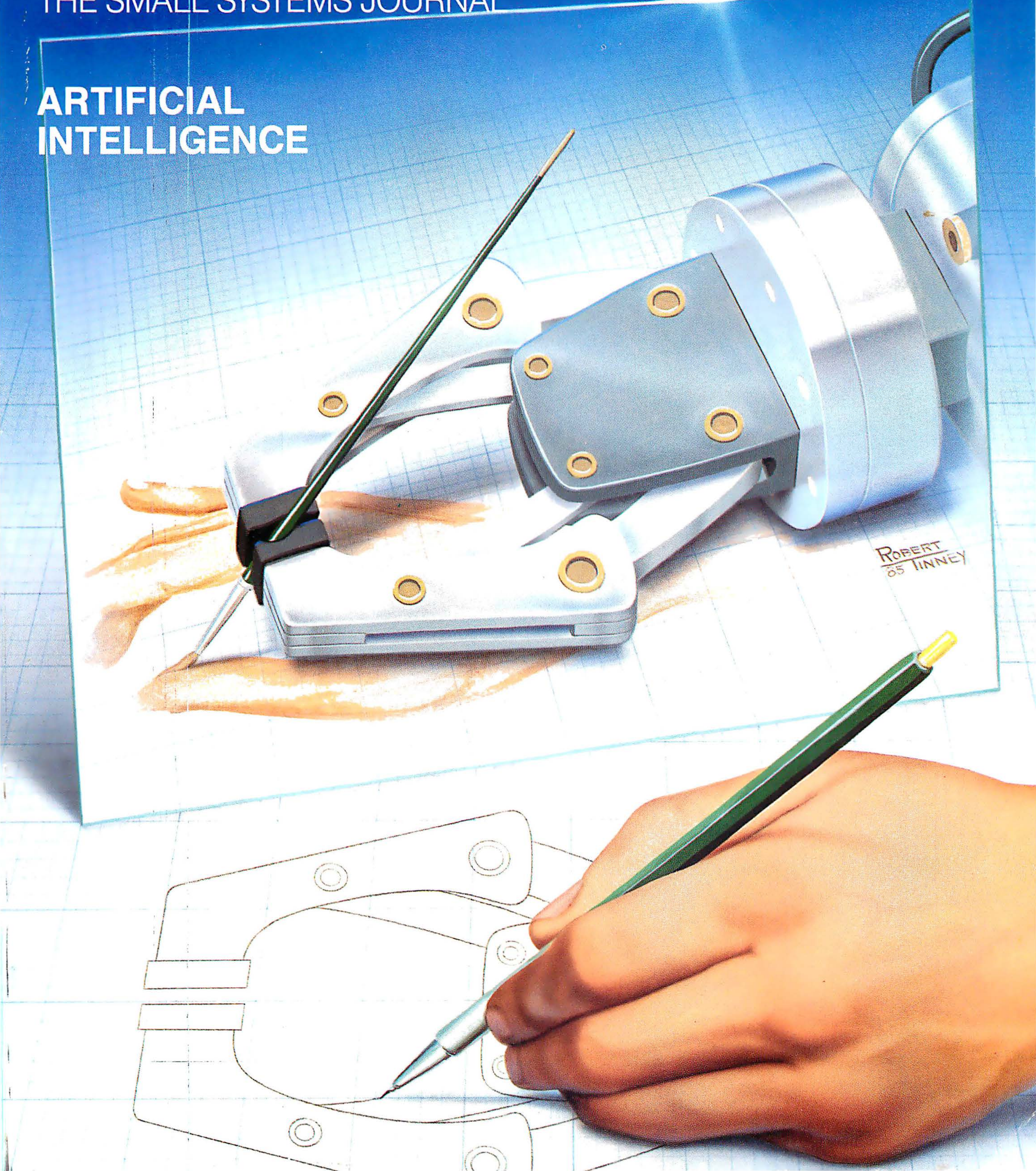
BYTE

THE SMALL SYSTEMS JOURNAL

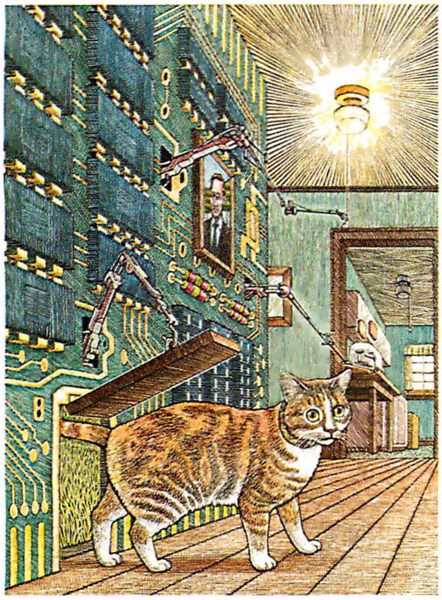
APRIL 1985 VOL. 10, NO. 4

$3.50 IN UNITED STATES
$4.25 IN CANADA / £2.10 IN U.K.
A McGRAW-HILL PUBLICATION
0360-5280

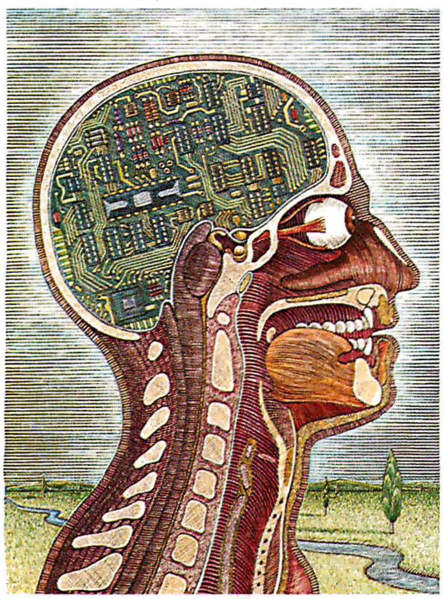
ARTIFICIAL INTELLIGENCE



C·O·N·T·E·N·T·S



100



124

FEATURES

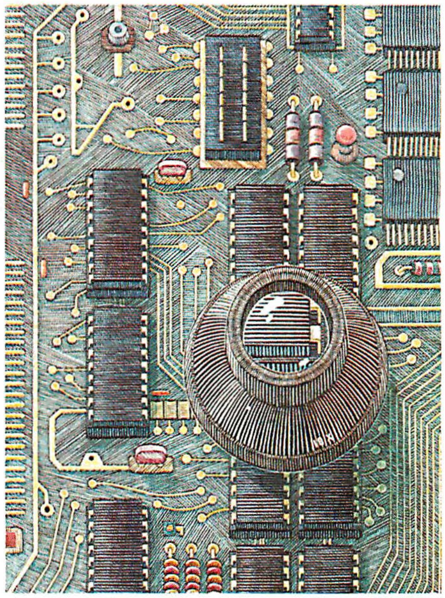
THEMES

BYTE (ISSN 0360-5280) is published monthly by McGraw-Hill Inc. Founder: James H. McGraw (1860-1948). Executive, editorial, circulation, and advertising offices: 70 Main St., Peterborough, NH 03458, phone (603) 924-9281. Office hours: Mon—Thur 8:30 AM — 4:30 PM, Friday 8:30 AM — 1:00 PM, Eastern Time. Address subscriptions to BYTE Subscriptions, POB 590, Martinsville, NJ 08836. Postmaster: send address changes, USPS Form 3579, undeliverable copies, and fulfillment questions to BYTE Subscriptions, POB 596, Martinsville, NJ 08836. Second-class postage paid at Peterborough, NH 03458 and additional mailing offices. Postage paid at Winnipeg, Manitoba. Registration number 9321. Subscriptions are $21 for one year, $38 for two years, and $55 for three years in the USA and its possessions. In Canada and Mexico, $23 for one year, $42 for two years, $61 for three years. $69 for one year air delivery to Europe. 17,100 yen for one year surface delivery to Japan. $37 surface delivery elsewhere. Air delivery to selected areas at additional rates upon request. Single copy price is $3.50 in the USA and its possessions, $3.95 in Canada and Mexico, $4.50 in Europe, and $5 elsewhere. Foreign subscriptions and sales should be remitted in United States funds drawn on a U.S. bank. Please allow six to eight weeks for delivery of first issue. Printed in the United States of America.

COVER ILLUSTRATION BY ROBERT TINNEY

BYTE April

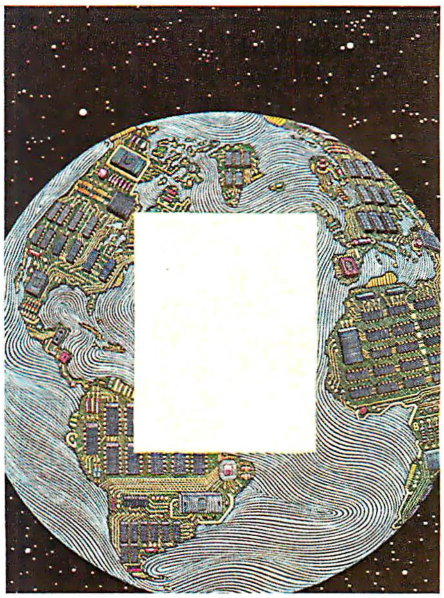
VOLUME 10, NUMBER 4, 1985



334

REVIEWS

KERNEL



353

Address all editorial correspondence to the Editor, BYTE, POB 372, Hancock, NH 03449. Unacceptable manuscripts will be returned if accompanied by sufficient first-class postage. Not responsible for lost manuscripts or photos. Opinions expressed by the authors are not necessarily those of BYTE.

Subscription questions or problems should be addressed to: BYTE Subscriber Service, POB 328, Hancock, NH 03449

SECTION ART BY DOUGLAS SMITH

THE TECHNOLOGY OF EXPERT SYSTEMS

BY ROBERT H. MICHAELSEN, DONALD MICHIE, AND ALBERT BOULANGER

Transplanting expert knowledge to machines

THE PURPOSE OF this article is to introduce expert systems. Initially, we'll define these systems. Next, we'll discuss methods for building them, including the advantages and disadvantages of each method. Finally, we'll review the computer resources needed to build and run expert systems.

DEFINITION

Expert systems are a class of computer programs that can advise, analyze, categorize, communicate, consult, design, diagnose, explain, explore, forecast, form concepts, identify, interpret, justify, learn, manage, monitor, plan, present, retrieve, schedule, test, and tutor. They address problems normally thought to require human specialists for their solution. Some of these programs have achieved expert levels of performance on the problems for which they were designed (see reference 6).

Expert systems are usually developed with the help of human experts who solve specific problems and reveal their thought processes as they proceed. If this process of protocol analysis is successful, the computer program based on this analysis will be able to solve the narrowly defined problems as well as an expert. (For a discussion of successful expert systems, see reference 2.)

Experts typically solve problems that are unstructured and ill-defined, usually in a setting that involves diagnosis or planning. They cope with this lack of structure by employing heuristics, which are the rules of thumb that people use to solve problems when a lack of time or understanding prevents an analysis of all the parameters involved. Likewise, expert systems employ programmed heuristics to solve problems. Figure 1 is an example of a complex heuristic used by TAXADVISOR, an expert system that gives estate-planning advice (see reference 17).

Experts engage in several different problem-solving activities. For instance, the following problem-solving activities have been identified in MYCIN (see figure 2): identify the problem, process data, generate questions, collect information, establish hypothesis space, group and differentiate, pursue and test hypothesis, explore and refine, ask general questions, and make a decision (see reference 11).

Experts are capable of

- Applying their expertise to the solution of problems in an efficient manner. They are able to employ plausible inference and reasoning from incomplete or uncertain data.
- Explaining and justifying what they do.
- Communicating well with other ex-

(continued)

Robert H. Michaelsen is an assistant professor of accounting at the University of Nebraska (Lincoln, NE 68588-0488). He received his Ph.D. in accountancy from the University of Illinois. Donald Michie is Director of Research at the Turing Institute (36 North Hanover St., Glasgow G1 2AD, Scotland). Formerly a professor at the University of Edinburgh, he is the author of numerous books and articles on artificial intelligence. Albert Boulanger is a scientist for Bolt Beranek and Newman Inc. (10 Moulton St., Cambridge, MA 02238). He has a master's degree in computer science from the University of Illinois at Urbana-Champaign.

perts and acquiring new knowledge.

- Restructuring and reorganizing knowledge.
- Breaking rules. They have almost as many exceptions as they have rules. They understand both the spirit and the letter of a rule.
- Determining relevance. They know when a problem is outside their expertise and when to make referrals.
- Degrading gracefully. At the boundaries of their expertise, they become gradually less proficient at solving problems, rather than suddenly incapable (see reference 4).

Expert systems have modeled only the first three expert capabilities to any extent, and even explanation and knowledge acquisition have just begun.

Expert systems, like human experts, can have both deep and surface representations of knowledge. Deep representations are causal models, categories, abstractions, and analogies. In such cases, we try to represent an understanding of structure and function. Surface representations are often empirical associations but are sometimes "compiled" from an understanding of structure and function. In the former case, the association between premises and conclusions of rules is based on empirical observation of past association. Causality is implicit in the rule, rather than explicit.

Deep representations enhance the explanatory powers of expert systems. With surface representations, all the system knows is that an empirical association exists; it is unable to explain why, beyond repeating the association. Where more fundamental insight is available, deep representation will enable the system to respond more substantively. If computer induction is used for knowledge acquisition, a model for understanding events in the domain (a deep representation) often guides the induction of rules from examples by distinguishing meaningful hypotheses from coincidences in the data. It is also likely that deep representation will enhance the incorporation of the last four previously listed expert capabilities into expert systems. Surface representations have offered little in this regard.

However, surface representations have their advantages if the only concern is problem-solving performance, empirical associations, or compiled understanding. They should be less costly to formulate than causal models. This lower cost can provide a reasonable level of explanation along with a primitive form of knowledge acquisition. If a domain's expertise is based on empirical association, as in many areas of medicine, surface representations are the only kind available (see reference 4).

The best approach to expert-system building is probably to use deep representations when they are cost-effective and surface representations for the rest of the system. This approach has already been explicated in a paper by Hart (reference 12) and implemented in Digitalis Advisor, a system that provided advice on digitalis dosages for cardiac patients (see reference 29).

RULE 216
(This rule applies to clients and is tried to find out whether a short-term trust should be recommended.)

If:
1) The client and/or spouse do wish to shift property income to another (not for legal support), etc., for at least 10 years or until the death of the beneficiary,
2) The client and/or spouse do desire to eventually reclaim control of this property (for retirement, estate liquidity, etc.),
3) The client and/or spouse are in a higher income bracket than the beneficiary,
4) The client and/or spouse are willing to relinquish control of the beneficial enjoyment of the property,
5) The client and/or spouse are able to provide for their living needs without this income, even in the event of disability or unemployment,
6) The client and/or spouse do not plan to have the trust income used to pay life-insurance premiums on his/her life without the consent of an adverse party,
7) The client and/or spouse do not plan to use the trust for a leaseback of assets, and
8) A: The client and/or spouse have a person (e.g., a parent) they are supporting without legal obligation with this property income (will lose a dependent if trust is formed),
   B: The client and/or spouse have a child, not a minor, that they will be putting through college with this property income (can set up early and accumulate income without tax problems), or
   C: The client and/or spouse are using some of their after-tax income for the benefit of some other taxpayer (child's marriage and/or home purchase, etc.),

Then: It is definite (1.0) that client should TRANSFER ASSETS TO A SHORT-TERM TRUST.

Figure 1: *An example of a* TAXADVISOR *rule.*

Building Expert Systems

An expert system is able to make decisions on a par with an expert primarily because its structure reflects the manner in which human specialists arrange and make inferences from their knowledge of the subject. The system is driven by a database of inexact and judgmental knowledge that is typically made up of if-then rules when surface representation is used, or frames and semantic nets when deep representation is used (see "A Glossary of Artificial Intelligence Terms" on page 138). Domain knowledge is processed in a strict order of deductive inference and is invoked by a pattern match with specified features of the task environment. Figure 3 is an example of pattern matching by TAXADVISOR. Because uncertainty is usually involved in expert judgments, expert systems must allow conclusions to be reached with less than certainty. Figure 4 illustrates how TAXADVISOR copes with uncertainty during a consultation. (For more information on uncertainty mechanisms in expert systems, see reference 32.)

The type of computer program that is used to develop an expert system cannot have its flow of control and data utilization rigidly fixed because such a structure is ill-adapted for simulating a human's responses to a complex, rapidly changing, and unfamiliar environment. Instead, such a program must examine the state of the world at each step of the decision process and react appropriately because new stimuli continually arise. The type of program that has been developed to cope with this constant change is a loosely organized collection of pattern-directed modules (PDMs) that detect situations and respond to them (see reference 31). The rule in figure 1 is a PDM from TAXADVISOR.

Each PDM examines and modifies data structures that model critical aspects of the external environment. In TAXADVISOR, the client's financial-planning situation and objectives constitute the environment. A PDM should be written as a single and separate unit that is independently meaningful within the task domain of the program. This aids incremental program growth and debugging, since revision of one PDM does not affect the others. It also provides explanation power; a single PDM can be used to explain a recommendation by the system.

Any system composed of several PDMs, one or more data structures that may be examined and modified by the PDMs, and an executive program to schedule and run the PDMs is called a pattern-directed inference system (PDIS). In effect, a PDIS factors complex problems into manageable, largely independent subproblems.

SURFACE REPRESENTATIONS

Rule-based systems (RBSs) were originally used in cognitive modeling of short-term memory. Since expert

(continued)

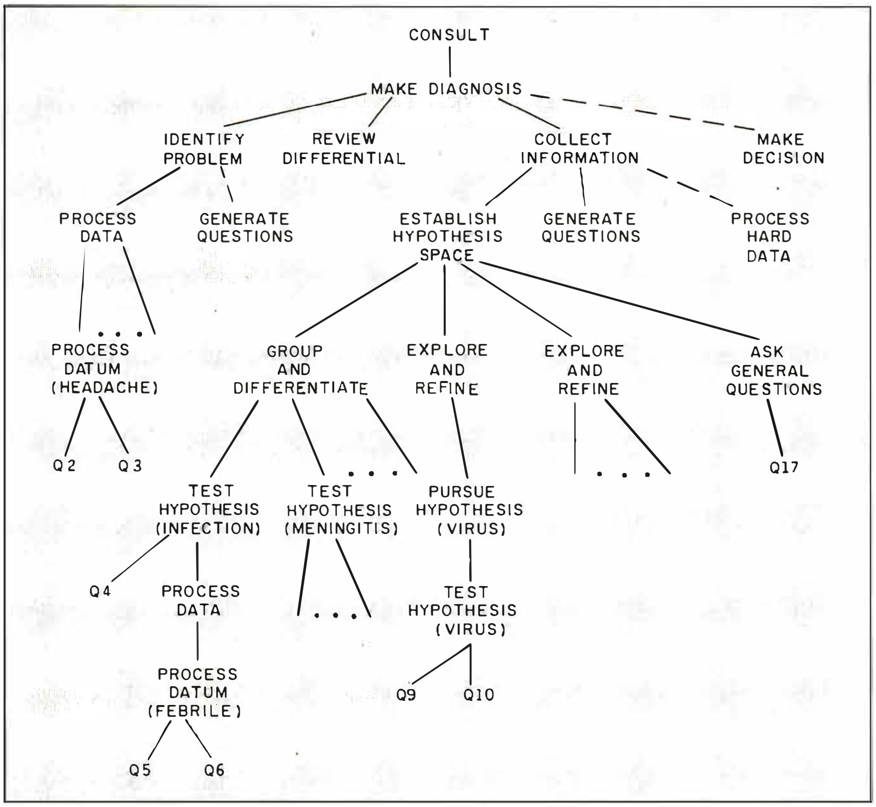


Figure 2: *The* MYCIN *problem-solving hierarchy. Question numbers* (Q2, *etc.*) *correspond to questions asked in the consultation. Solid lines show tasks actually done, dashed lines those that might be done.* (*Figure used with permission; see reference* 11.)

Necessary Attributes for Short-Term Trust	Prestored Value	Client's Value
1) Does client wish to shift property income to another (not for legal support) for at least 10 years or until the death of the beneficiary?	yes	yes
2) Does client desire to eventually reclaim control of the property?	yes	yes
3) Is client in a higher income-tax bracket than the beneficiary?	yes	yes
4) Is client willing to relinquish control of the beneficial enjoyment of the property?	yes	yes
5) Is client able to provide for his living needs without this income even if disabled or unemployed?	yes	yes
6) Does client plan to have trust income pay life-insurance premiums on his life without consent of an adverse party?	no	no
7) Does client plan to use the trust for a leaseback of assets?	no	no
8) Does client have a person he is supporting without legal obligation?	yes	yes

Figure 3: *An example of pattern matching done by* TAXADVISOR, *performed to determine if the client should be forming a short-term trust. The Prestored Value column shows the pattern of attribute values that a client must have before* TAXADVISOR *will recommend a transfer of assets to a short-term trust. Since the client's pattern matches the prestored one, the trust will be recommended.*

(begin)

—Formation of a Temporary Trust—

1) Does client wish to shift property income to another (not for legal support) for at least 10 years or until the death of the beneficiary?
**YES
2) Does client desire to eventually reclaim control of the property?
**YES
3) Is client in a higher income-tax bracket than the beneficiary?
**YES
4) Is client willing to relinquish control of the beneficial enjoyment of the property?
**WHY

This will aid in determining whether client should form a temporary trust. It has already been established that

1) Client wishes to shift property income to another (not for legal support) for at least 10 years or until the death of the beneficiary,
2) desires to eventually reclaim control of the property, and
3) is in a higher income-tax bracket than the beneficiary.

Therefore, if

1) client is willing to relinquish control of the beneficial enjoyment of the property,
2) is able to provide for his living needs without this income, even if disabled or unemployed,
3) does not plan to have trust income pay life-insurance premiums on his life without the consent of an adverse party,
4) does not plan to use the trust for a leaseback of assets, and
5) A: has a person (e.g., a parent) he is supporting without legal obligation with this property income,
B: has a child, not a minor, that he will be putting through college with this property income, or
C: is using some of his after-tax income for the benefit of some other taxpayer,

then

client should form a temporary trust
(back to question 4 . . .)

**YES

5) Is client able to provide for his living needs without this income, even if disabled or unemployed?
**YES (8) [Whenever a response is made with less than certainty, the system user enters a number betwen 1 and 9 indicating his degree of certainty in that response.]
6) Does client plan to have trust income pay life-insurance premiums on his life without consent of an adverse party?
**NO
7) Does client plan to use the trust for a leaseback of assets?
**NO
8) Does client have a person he is supporting without legal obligation?
**YES
I recommend that the client form a short-term trust.
[The degree of certainty that the system has in this recommendation is .8. This certainty factor (CF) was calculated as follows. The temporary trust rule's action CF was 1.0 and it had an "AND" premise. In such a case, the rule's CF is the minimum CF used in the responses, or .8. Since the system's threshold CF is .2, the recommendation was made.]

(end)

Figure 4: *A partial interactive consultation with* TAXADVISOR. *The user's input is in uppercase.*

systems attempt to imitate people, it was natural that RBSs would also be used in their development. To date, RBSs are by far the most common structure for expert systems. Among the successful rule-based expert systems that have been developed are the following:

- MYCIN—diagnoses infections (reference 26)
- HEURISTIC DENDRAL—identifies organic compounds (reference 8)
- PROSPECTOR—aids geologists in evaluating mineral sites (reference 5)
- PUFF—analyzes pulmonary function tests (reference 15)
- INTERNIST—performs diagnosis in internal medicine (reference 22)
- XCON (formerly R1)—configures the VAX-11/780 computer system (reference 16)
- SACON—provides engineers with advice on structural analysis (reference 1)

Because of the popularity of RBSs, several domain-independent systems have been developed to make it much easier to build rule-based expert systems in many fields. The following is a partial list of domain-independent systems:

- EMYCIN (reference 30)
- AGE (reference 21)
- OPS5 (reference 16)
- ADVISE (reference 18)
- Hearsay-3 (reference 7)
- AL/X (reference 23)
- Expert-Ease (Human Edge Software Corp., Palo Alto, California)
- KS 300 (Revamped EMYCIN; from Teknowledge Inc., Palo Alto, California)
- KES (Intelligenetics Inc., Palo Alto, California)
- Personal Consultant (Texas Instruments Inc., Dallas, Texas)

An RBS is composed of PDMs called rules, each with a left-hand side (the antecedent, a logical combination of propositions about the database) and a separate right-hand side (the consequent, a collection of actions). An RBS separates data examination (done by the left-hand side) from data modification (done by the right-hand side of the rule).

Most RBSs are production systems (PSs), in which matching and scheduling are explicitly defined by the operation of the executive (control) program. The control schema can be characterized as having four basic parts:

1. Selection: select relevant rules and data elements. Selection may be trivial (e.g., on each cycle all rules and all data elements can be considered) or quite complex (e.g., special filters can be designed to eliminate from consideration many rules that could not possibly match the current data). In TAXADVISOR, rules are organized in a hierarchy to narrow the rules considered.
2. Matching: compare active rules against active data elements, looking for patterns that match, i.e., rules whose conditions are satisfied. Figure 3 is an example of pattern matching.
3. Scheduling: decide which "satisfied" rule should be "fired." "Firing" consists of accessing and executing the procedures associated with the pattern elements that matched the current data. If more than one rule is satisfied, conflict-resolution heuristics are used to decide which rule to fire.
4. Execution: fire the rule chosen during the scheduling process. The result of execution is a modification of data elements or structure. With TAXADVISOR, execution results in an estate-planning recommendation for a client. This is illustrated in the test consultation in Figure 4 (see reference 31).

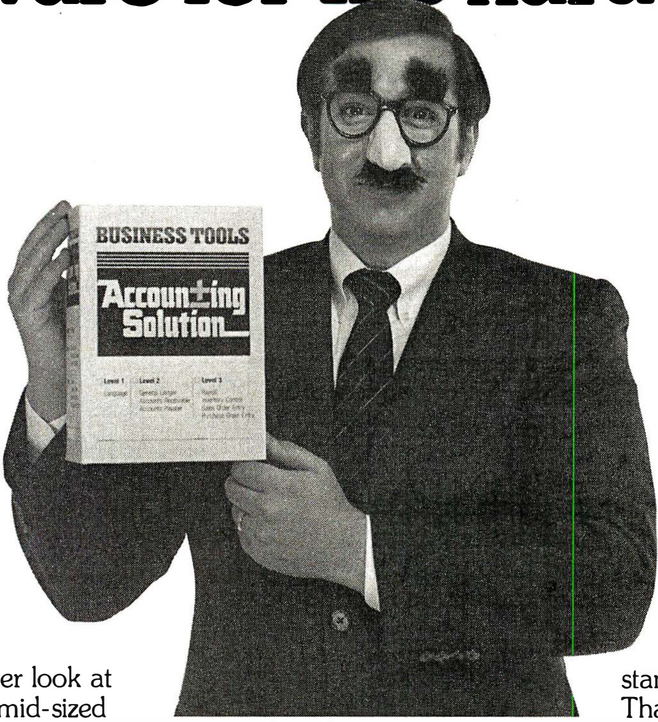
PSs are either consequent-driven systems or antecedent-driven systems. A consequent-driven (backward-chaining) system, which is the type used in TAXADVISOR, uses rule consequents (which represent goals) to guide the search for rules to fire (with TAXADVISOR, estate-planning actions to recommend). The system collects those rules that can satisfy the goal in question and tries to satisfy the consequents of those rules, which usually represent the values of variables. In order to find these values, the values of the rule antecedent must be found. To satisfy each antecedent, which represents a subgoal, the system collects those rules whose consequents satisfy its value. The process of working backward through the rules from consequents to antecedents to consequents in search of a causal chain that will satisfy the goal is called backward chaining. (For a simple backward-chaining program written in BASIC, see "Knowledge-Based Expert Systems Come of Age" by Richard O. Duda and John G. Gaschnig, September 1981 BYTE, page 238.)

With antecedent-driven (forward-chaining) systems, program execution consists solely of a continuous sequence of cycles terminating when a rule's action dictates a halt. At each cycle, the system scans the antecedents and determines all rules with antecedents that are satisfied by the contents of the database. If there is more than one such rule, select one by means of a conflict-resolution strategy. Perform all actions associated with the selected rule and change the database accordingly. For example, with R1 (XCON), you enter all the information on the problem into the database, and the system then applies the rules to reason forward from the data to the conclusions. In summary, forward chaining consists of putting the rules in a queue and then using a recognize-act cycle on them.

Some forward-chaining systems try to control the search for rules in the recognize cycle by grouping rules into packets. These rule groupings are appealing conceptual structures, since they group rules according to the subtopic that they deal with. Object-oriented programming can also be used to organize collections of rules. In object-oriented programming, we give objects behavior, and thus we can distribute the control of rules into rule, rule-packet, and domain objects. This approach, which has been taken in LOOPS, a domain-independent system (see reference 27), also allows multiple instantiations of the same set of rules to solve subproblems of the

(continued)

Finally, business computer software for the hard-nosed.



No one takes a harder look at software than small to mid-sized businesses.

So take a long, hard look at The Accounting Solution,™ a new, totally integrated software package from Business Tools, Inc.™

You'll find its breakthrough features are designed to pay off where it counts—on the bottom line.

Hard-nosed economy, $99.*

Contrary to popular opinion, you don't need a small business loan to buy quality software. Not if you're buying The Accounting Solution. For $99, you get a language/data base manager with more hard-nose capabilities and speed than any program available at any price; $249 buys the language plus accounts receivable/payable and general ledger; $399 gets you all the above plus inventory control, sales order entry, purchase order entry and payroll. Even more good news for the budget minded—source code is included with applications.

Easy for any hard-nose.

The Accounting Solution is easy enough for the novice hard-nose to use within minutes of receiving the package. Yet it's also sophisticated, offering unlimited flexibility and opportunity to the hot-shot hard-nose. And it's designed to run on CP/M-80, MP/M-80, IBM PC and compatibles.**

Multi hard-nose capability.

The Accounting Solution never stands in the way of progress. Thanks to multi-user capability, two or more hard-noses can use the same application at the same time.

Hard-nosed flexibility.

With The Accounting Solution, it's easy to change your mind because the source code is so simple to modify. Ready to grow? Great. You can change hardware without spending a dime on new software.

Take it from hard-nose Phil Mickelson.

Phil created The Sensible Solution,** a highly respected software package. Now he's offering the next step, another breakthrough: The Accounting Solution. It's simple. Sophisticated. Affordable. And backed by Phil's reputation and personal service. If you're looking for hard-nosed value and quality, you'll agree, The Accounting Solution is the only solution.

BUSINESS TOOLS™

Accounting Solution

Write or call:

Business Tools, Inc.
4038-B 128th Avenue SE
Bellevue, WA 98006

1-800-648-6258

Washington State:
(206) 644-2015

Dealer inquiries welcome.

*Suggested retail price.
**CP/M-80 and MP/M-80 are registered trademarks of Digital Research, Inc.; IBM PC is a registered trademark of International Business Machines Corporation; The Sensible Solution trademark rights are claimed by O'Hanlon Computer Systems.*

same type within one session.

The primary difference between backward and forward chaining is a top-down versus bottom-up style of linking rules together. Though the most common, these are not the only control structures for rule-based systems. For example, rules are represented as an "inference" network in PROSPECTOR (see reference 5).

DEEP REPRESENTATIONS

Frame- and network-based approaches allow the implementation of "deeper-level" reasoning such as abstraction and analogy. Reasoning by abstraction and analogy is an important expert activity. You can also represent the *objects* (e.g., "pump" in figure 5) and *processes* (e.g., the "start" instructions in figure 5) of the domain of expertise at this level. What is important are the *relations between objects*. Deep-representation expert systems perform inference using relations represented by networks or frames. A semantic network is a graph of the relations. A frame or script system (see references 20 and 24) organize the objects and their relations into *entities* (recognizable collections of objects). Frame systems also provide a system to inherit attributes from a taxonomy of entities. Thus, a frame system implements the semantics of some of the relations between objects. With a semantic-net or frame system you can represent objects of the domain of expertise as well as the process, strategies, etc., that are also part of the domain. The control of frame or semantic-net systems is usually much more involved than with surface systems and is implemented in a way that an explanation facility can't get at. But surface systems are "shallow"; a surface system may be viewed as a projection of deep-level knowledge of a domain for a specific

(continued)

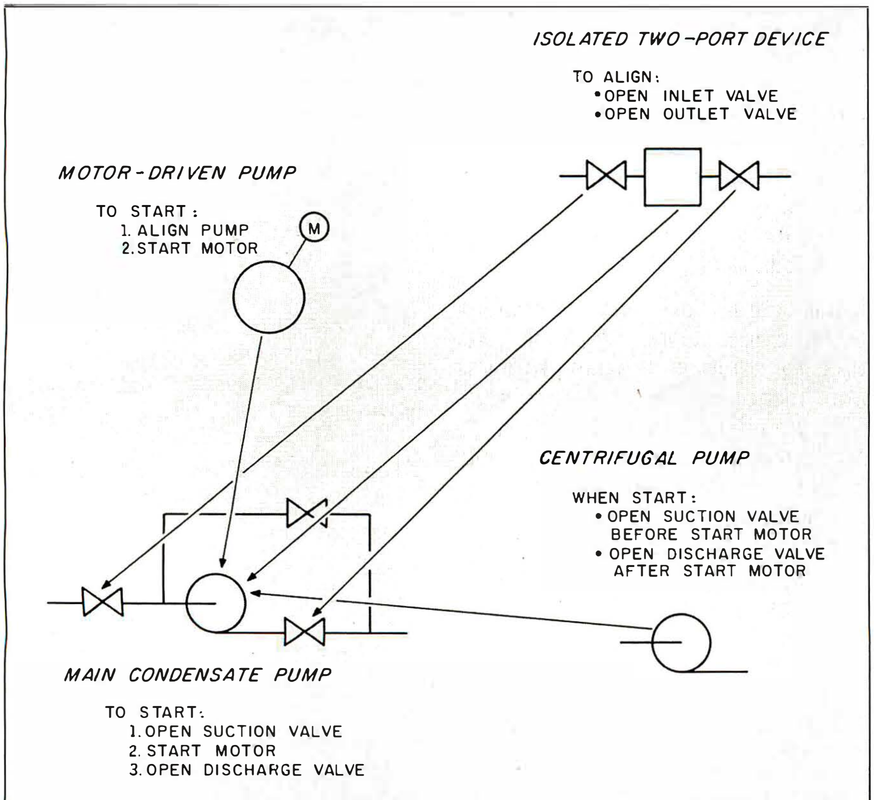


Figure 5: *Procedure steps are obtained from the subcomponents and abstractions of an object, here a main condensate pump. This example comes from Steamer, a tutorial system designed to teach operating procedures of shipboard steam plants.*

Some systems have a built-in capability for taking a file of expert decisions and generalizing from this knowledge an executable rule.

expert activity.

One type of expertise that has been represented with a deep-level approach is tutoring (see "The LISP Tutor" by John R. Anderson and Brian J. Reiser on page 159). Here we want to convey to the pupil domain knowledge that is best represented at the deep level: concepts, abstractions, analogies, and problem-solving strategies.

Steamer is a training aid developed jointly by Bolt Beranek and Newman Inc. and the Navy Personnel Research and Development Center. Its goal is to teach operating procedures of shipboard steam plants. These procedures consist of a series of steps on subcomponents of the plant. The components and procedures are represented as frames in Steamer, as are the abstractions of components and procedures that experts use in teaching steam-plant operations. The steps of a procedure come from the abstractions and subcomponents of the device the procedure applies to. The ordering of the steps comes from a third represented entity: operating principles. These principles are culled from experienced operators and represent "compiled" knowledge of steam-plant operation (although they are not represented as rules but frames).

Knowledge Acquisition

The following are ways of acquiring knowledge in a form that can be used by an expert system (reference 19):

- being told
- analogy
- example
- observation, discovery, and experimentation
- reasoning from deep structure

The manual acquisition of knowledge from human experts is a very labor-intensive process. There is an acknowledged need to have aids for knowledge acquisition as part of the system.

Methods to speed knowledge acquisition are now becoming available in the form of machine learning of rules from examples. Systems such as Expert-Ease have a built-in capability for taking a file of expert decisions from you and generalizing from these an *executable rule*. In a sense, you are able to transplant chunks of decision-making skill from your own brain to the personal computer, a possibility foreseen as early as 1966 by Earl Hunt and his colleagues.

The machine procedure that allows this skill transplant was developed from a Pascal-coded program called ID3 (Iterative Dichotomiser 3) due to Professor Ross Quinlan of the New South Wales Institute of Science and Technology.

A number of conclusions follow from Quinlan's work:

1. It is possible, using such a program, to generate machine-executable solutions for complex decision problems in a fraction of the time a programmer would need for developing a solution by conventional hand coding.
2. The resulting solutions are super-efficient as compared with those obtainable by the old hand methods.
3. It is important to make up your mind in advance whether super-efficiency is all you demand of a machine-executable solution, or whether you *also* want the resulting rule base to be understandable on inspection.

If the answer to the third statement above is that user transparency of induced rules is desired, then (unless it is a very small one) do *not* treat your problem as one big superproblem with a single associated file of examples. Instead, first break it down into a main problem and a set of subproblems, even going further (to the level of sub-subproblems) if the complexity of the problem domain seems to call for it. The originators of this style, which is known as "structured induction," are Drs. Shapiro and Niblett (reference 25). Corporations enjoying the use of powerful inductive generators such as ITL's FORTRAN-based EX-TRAN system or Radian Corporation's C-coded RuleMaster have applied the approach to the building of complex systems for trouble-shooting large transformers, severe-storm warning, circuit-board fault diagnosis, and user-friendly guidance to set up numerical batch jobs in seismic analysis in the oil industry. Rates of production of compact installed code in excess of 100 lines per worker day are now commonly reported.

Any robust expert system takes a tremendous amount of resources to develop. Once developed, the knowledge along with the control structure can be "compiled out"; that is, the system of rules is rewritten into a piece of code that performs the same function on a personal computer. For example, some expert systems (ADVISE, EMYCIN, OPS5—see reference 10) can generate code or other primitive forms of the knowledge for use on a personal computer. (Systems run on a personal computer are usually referred to as "delivery systems.")

Knowledge Representation

As AI researchers point out, a robust expert system that can explain, justify, acquire new knowledge, adapt, break rules, determine relevance, and degrade gracefully will have to use a multitude of knowledge representations that lie in a space whose dimensions include deep/surface, qualitative/quantitative, approximate (uncertain)/exact (certain), specific/general, and descriptive/prescriptive. Systems that use knowledge represented in different forms have been termed *multilevel systems*. Steamer is an example of one such expert system.

Steamer uses the following representations:

1. A graphical (icon) representation of the objects of the Steamer domain, such as valves, pumps, tanks, and systems composed of these.
2. A frame representation of Steamer objects, procedures, and operating principles. This is used for describing, explaining, categorizing, abstracting, and referring.
3. An assertional database where assertions about Steamer entities can be made and retracted.
4. A quantitative numerical simulation of the steam plant that is used in illustrating cause and effect and ramifications of the application (or misapplication) of procedures.

Work is just beginning in building such multilevel systems, and they will be a major research topic for this decade. Work needs to be done in studying and representing in a general way the different problem-solving activities an expert does (see reference 3). When you build expert systems, you realize that the power behind them is that they provide a regimen for experts to crystallize and codify their knowledge, and in the knowledge lies the power.

Resources Needed

Before resource needs are discussed, you must precisely define the type of expert system you want to build. If you wish to build a large, "custom" model expert system (i.e., it is not feasible to use many of the smaller domain-independent systems that are available), you will need substantial resources: large memory, good language support, and fast execution of the code. You may need to develop such a system in LISP on hardware specialized to processing the language, or on time-sharing machines with a large address space. Such "custom" systems are usually referred to as "prototype" or "development" systems. They can either be developed for a specific domain (e.g., MYCIN) or be domain-independent (e.g., ADVISE).

If you are able to build a less complex expert system using an existing domain-independent system or if the system has a rule-compilation facility that allows applications to be run on personal computers, then a personal computer (preferably with 512K bytes) is sufficient. If all you need are resources to run an existing expert system, a large personal computer should nearly always be sufficient.

There is no obvious line of demarcation for a given project. However, certain barriers make personal computer use less desirable as system size and complexity increase.

System Barriers

Many high-level languages do not offer the right primitives (i.e., programming-language statements) for developing expert systems. Among the desirable primitives are

- A parser or interpreter that parses statements during program run time. Without this, you have to write a parser for the rules.
- List and nonnumeric processing primitives.
- A language design that allows incremental compilation and other fast prototyping facilities. Incremental compilation enables you to recompile a function or other portion of a file without recompiling the entire file.

The view that many people in the field are adopting is that high-level languages like Pascal, Ada, and C are acceptable for the delivery system, but for prototyping, a language like LISP or Prolog is preferred. Program-generation tools are then used to write the system in the delivery language.

The knowledge-intensive approach to expert systems implies that the memory will be highly utilized in all but the most nontrivial applications. AL/X is one example that ran on a 64K-byte machine, but it was a small expert-system shell. Since memory prices have gone down and many small machines have broken the 64K-byte barrier, we can expect that more expert systems can be developed, at least for the delivery system, on per-

(continued)

Some researchers predict that memory needs of advanced expert systems will drive development of encyclopedic memories.

sonal computers. Some researchers predict that the memory needs of advanced expert systems will drive the development of encyclopedic memories for personal computers.

CONCLUSION

Expert systems can be built in many ways, involving rules, networks, frames, and combinations thereof, with all sorts of variations within these categories with respect to knowledge representation and control. We could not begin to cover all possible approaches to building expert systems, since new ones are being developed almost daily.

Even if the most efficient approach has been ascertained for the domain in question, the most cost-effective computer resource must still be determined. In most cases, approach selection at least narrows the choice for resources; in some cases, approach and resources can be selected together. However, this hardly reduces the complexity of the choice. To make matters worse, computer resources are changing as rapidly as the new system-building approaches are being developed. The best we can hope to convey is an awareness of the opportunities and complexities involved in the development of expert systems. ■

REFERENCES

1. Bennett, J. S., and R. S. Englemore. "SACON: A Knowledge-Based Consultant for Structural Analysis." *IJCA*, 179, 1979, page 47.
2. Bramer, M. A. "A Survey and Critical Review of Expert Systems Research." *Introductory Readings in Expert Systems*, D. Michie, ed. London and New York: Gordon and Breach, 1982.
3. Chandrasekaran, B., and Sanjay Mittal. "Deep Versus Compiled Knowledge Approaches to Diagnostic Problem-Solving." *International Journal of Man-Machine Studies*, #19, 1983, page 425.
4. Davis, R. "Expert Systems: Where Are We? and Where Do We Go From Here?" *AI Magazine*, Spring 1982, page 3.
5. Duda, R., J. Gaschnig, and P. Hart. "Model Design in PROSPECTOR Consultant System for Mineral Exploration." *ESMA*, 1979, page 153.
6. Duda, R. O., and E. H. Shortliffe. "Expert Systems Research." *Science*, April 1983, page 261.
7. Erman, L. D., P. E. London, and S. F. Fickas. "The Design and Example Use of Hearsay 3." *Proceedings of IJCA* no. 7, 1981, page 409.
8. Feigenbaum, E. A., B. G. Buchanan, and J. Lederberg. "On Generality and Problem Solving: A Case Study Using the DENDRAL Program." *Machine Intelligence 6*, B. Meltzer and D. Michie, eds. New York: Edinburgh University Press and Halsted Press (Wiley), 1971, page 165.
9. Forbus, Kenneth D. "Qualitative Process Theory." *MIT Technical Report 789*, MIT AI Laboratory, May 1984.
10. Forgey, C. L. "Rete: A Fast Algorithm for the Many Pattern/Many Object Match Problem." *Artificial Intelligence*, September 1982.
11. Hasling, Diane Warner, William J. Clancey, and Glenn Rennels. "Strategic Explanations for a Diagnostic Consultation System." *International Journal of Man-Machine Studies*, January 1984, page 3.
12. Hart, P. "Direction for AI in the 80's." *SIGART Newsletter*, November 1981, page 11.
13. Hollan, James, Edwin Hutchins, and L. Weitzman. "Steamer: An Interactive Inspectable Simulation-Based Training System." *AI Magazine*, Summer 1984, page 15.
14. Hutchins, Edwin, Terry Roe, and James Hollan. "Project STEAMER: VII. A Computer-Based System for Monitoring the Boiler Light-Off Procedure for a 1078-Class Frigate." *NPRDC Technical Note* 82-85, August 1982.
15. Kunz, J. C., et al. "A Physiological Rule-Based System for Interpreting Pulmonary Function Tests." Heuristic Programming Project, Memo HPP-78-19, Stanford University, 1978.
16. McDermott, J. "R1: A Rule-Based Configuror of Computer Systems." Computer Science Department, Carnegie-Mellon University, 1980.
17. Michaelsen, R. H. "An Expert System for Federal Tax Planning." *Expert Systems: The International Journal of Knowledge Engineering*, October, 1984, page 149.
18. Michalski, R. S., A. B. Baskin, A. Boulanger, R. Reinke, L. Rodewald, M. Seyler, K. Spachman, and C. Uhrik. "A Technical Description of the ADVISE Meta Expert System." Department of Computer Science, University of Illinois at Urbana-Champaign, 1983.
19. Michalski, R. S., J. Carbonell, and T. Mitchell, eds. *Machine Learning: An Artificial Intelligence Approach*. Los Altos, CA: Tioga Publishing Company, 1983.
20. Minsky, M. "A Framework for Representing Knowledge." *The Psychology of Computer Vision*, P. Winston, ed. New York: McGraw-Hill, 1975.
21. Nii, H. P., and N. Aiello. "AGE (Attempt to Generalize): A Knowledge-Based Program for Building Knowledge-Based Programs." *IJCA*, 179, 1979, page 645.
22. Pople, H. E., J. D. Myers, and R. A. Miller. "Dialog: A Model of Diagnostic Logic for Internal Medicine." *IJCA*, 175, 1975, page 848.
23. Reiter, J. "AL/X: An Expert System Using Plausible Inference." Intelligent Terminals Ltd., University of Edinburgh, 1980.
24. Schank, R. C., and R. P. Abelson. *Scripts, Plans, Goals, and Understanding*. Hillsdale, NJ: Larrence Erlbaum Associates, 1977.
25. Shapiro, A., and T. Niblett. "Automatic Induction of Classification Rules for a Chess Endgame." *Advances in Computer Chess 3*, M. R. B. Clarke, ed. Oxford: Pergamon, 1982.
26. Shortliffe, E. H. *Computer-Based Medical Consultations: MYCIN*. New York: American Elsevier/North-Holland, 1976.
27. Stefik, Mark, Daniel G. Bobrow, Sanjay Mittal, and Lynn Conway. "Knowledge Programming in LOOPS: Report on an Experimental Course." *AI Magazine*, Fall 1983.
28. Stevens, Albert, and Bruce Roberts. "Quantitative and Qualitative Simulation in Computer Base Training." *Journal of Computer Based Instruction*, volume 10, numbers 1 and 2, Summer 1983, page 16.
29. Swartout, W. R. "A Digitalis Therapy Advisor with Explanations." *Technical Report 176*, MIT Lab for Computer Science, February 1977.
30. Van Melle, W. "A Domain-Independent Production Rule System for Consultation Programs." *IJCA*, 179, 1979, page 923.
31. Waterman, D. A., and F. Hayes-Roth, eds. *Pattern-Directed Inference Systems*. New York: Academic Press, 1978.
32. Whalen, Thomas, and Brian Schott. "Issues in Fuzzy Production Systems." *International Journal of Man-Machine Studies*, #19, 1983, page 57.