Introduction to Expert Systems

Ware Myers
Contributing Editor

The line dividing intelligent from nonintelligent systems keeps shifting as computers and their software become more capable. When computers were new 35 years ago and could do little except arithmetic functions, some called them giant brains, implying intelligence. Today, we do not regard the giant brain's successor, the pocket calculator, as intelligent. In fact, we do not regard a payroll program of hundreds of thousands of lines of code, running on a mainframe now orders of magnitude more powerful than its ancestors, as intelligent.

There are current systems, however, that we do regard as intelligent. Expert systems are an example; they are a subset of intelligent systems.[1] Still, as the research community explores the problems that these programs address, the resulting systems will no doubt come to seem commonplace and hence something less than intelligent. The mystery of intelligence will continue to be a moving target.

What we mean by intelligence is broadly characterized by what human beings are able to do. As human beings we have an intuitive feel for what that is. Looking at the human brain from the outside, we have considerable knowledge of what it can do. From the perspective of inside-the-brain, however, we have taken only a few steps toward understanding how the neurons operate. We do not have enough knowledge at this level to create intelligent systems by simulating the detailed structures of the brain artificially. Still, in the 30 years since John McCarthy coined the term artificial intelligence, outside-the-brain approaches to intelligent systems, based on studying the nature of particular problems, have achieved some success. In the last few years, practical applications of expert systems have appeared.[2]

A working concept of what artificial intelligence embraces is found in the topics being studied by the artificial-intelligence community, summarized in Table 1. Many of these terms cannot be simply defined. For example, one book takes an entire chapter to define "expert system" and to set the boundaries of the concept.[3]

Fundamentals of expert systems

In Figure 1, we examine the relationships of one class of intelligent system, the expert system, to the environment in which it functions. On the right side of the figure, the expert system acquires knowledge through knowledge-acquisition software tools from a knowledge engineer who, in turn, acquires the knowledge from a domain expert. Domain refers to a specific field of knowledge. Once the expert system has been built, this relationship could be ended. On the other hand, since the domain normally continues to develop, the relationship usually continues.

On the left side, the expert system interfaces to the user. The user puts into the system facts and suppositions of varying degrees of probable truth and receives answers, recommendations, or diagnoses of some degree of reliability. The user usually works through a keyboard interface in a formal language. As natural-language capability becomes available, it can make this interchange more friendly. Eventually, speech recognition and generation may replace the keyboard.

At the bottom of the figure, a general database is shown interfacing to the expert system. At present, most expert systems depend upon a specialized knowledge base. They will become more usable with existing data processing systems when they become able to draw upon general databases, either in the immediate computer system or on a network.

Turning this relationship around, large, existing databases will become more useful when intelligence can be added to them. Query languages are beginning to add intelligence to the task of searching a database. At present, it may take an unreasonable amount of time and computer power to search an existing database. An intelligent front end capable of guided search could home in on information more quickly.

0885-9000/86/0200-0100$01.00 © 1986 IEEE

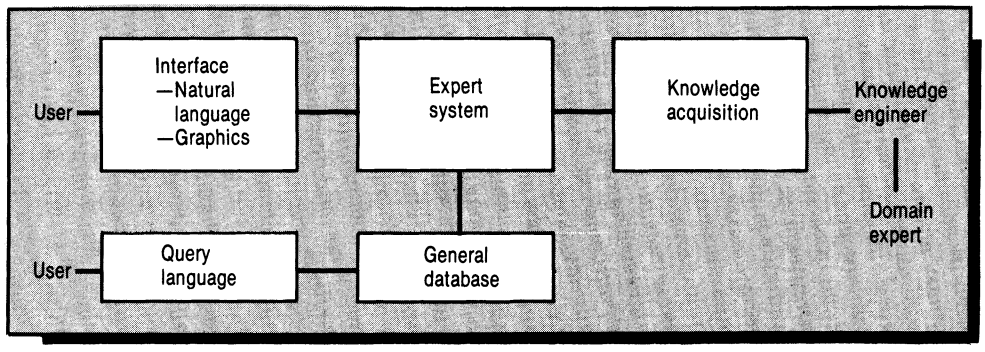


Figure 1. An expert system is related to the domain from which its knowledge is drawn, to the user for whom it solves problems, and eventually to a general database.

Expert system concept. In Figure 2, we divide the expert-system concept into a knowledge base and an inference engine. The knowledge base is unique to a particular domain, but the inference engine may be common to a number of domains that have similar characteristics. Thus, a number of inference engines were initially developed for particular applications, but were later separated from the knowledge specific to that application and used with other knowledge bases.

At the next level of detail, we identify four blocks: assertions, knowledge relationships, search strategy, and explanation tracing.

Assertions. This block, sometimes called the working memory or temporary data store, contains "declarative knowledge about the particular problem being solved and the current state of affairs in the attempt to solve the problem."[4] There are several ways to represent this data: first-order predicate logic, frames, and semantic networks.

Predicate logic represents the declaration Richard gave Jean a rose in the form *Give (Richard, Jean, rose).* A frame for this same information might be

name of frame: F1
type of frame: transfer of possession
source: Richard
destination: Jean
agent: Richard
object: rose

In the semantic network, each node contains an object and the lines between the nodes represent the relationships.

Knowledge relationships. This block contains formulas showing the relationship among several pieces of information. The most common formula is the production rule, such as

If it is clear and hot and muggy, then it is summer.

Here we have three antecedents connected by logic AND, which, when satisfied, lead to the consequence that it is

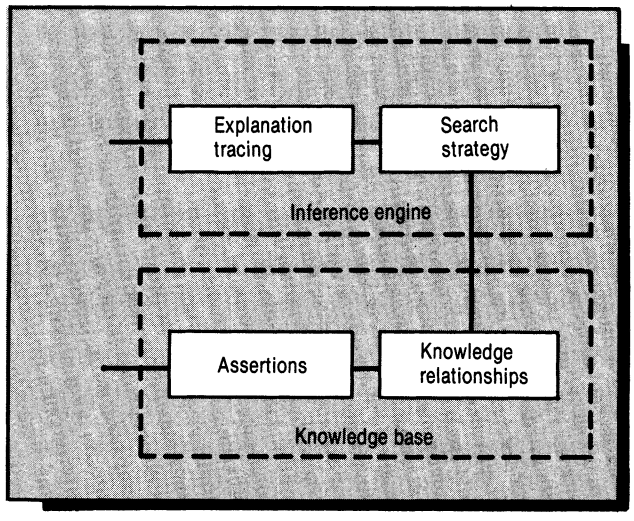


Figure 2. An expert system may be thought of as a knowledge base that an inference engine searches for answers to a particular problem, such as a diagnosis fitting a set of symptoms. The symptoms rest in the assertion block and are matched against symptom-disease relationships in the knowledge-relationships block under the control of the search strategy. The reasoning process by which the result was reached may be traced by the explanation capability.

Table 1. Session topics at recent artificial-intelligence conferences suggest the range of the field.

Automated reasoning	Logic programming, Prolog
Automatic programming	Machine learning
Cognitive modeling	Natural language, including speech
Expert systems	Perception, including visual,
Intelligent databases	auditory, tactile
Knowledge acquisition	Problem solving and search
Knowledge bases	Robotics
Knowledge representation	Theorem proving
Lisp programming	Vision

summer. In addition to antecedent-consequence, the two parts of an if-then production rule may be called a situation-action pair or a premise-conclusion pair. In the Lisp language, the relationship is called a clause and the clause has two parts: a test and a result. In Prolog, relationships are shown in the first-order predicate logic.

When all three of the antecedents, *clear, hot,* and *muggy,* are present as assertions, they match the left-hand side of the production rule, leading to the consequence, *it is summer.* In addition, there are two further steps to the making of matches: (1) The production rules may be listed in some order believed to facilitate the search for a match; and (2) the search-strategy block may be able to invoke a sequence that finds the match more quickly than random search does.

Search strategy. The simplest arrangement of the production rules is to list them in no particular order. With this arrangement, new rules may be tacked on, making it easy to grow the system as more is learned about the problem. Each asserted fact is then run through the production rules until the match is found. With a small number of rules, given the high speed of computer operation, it is practical to search a random list.

If the number of rules is large, they may be partitioned into sublists, or contexts, on some logical basis. The search strategy then uses a higher-level rule, or a metarule, based on the logic of the partition, to determine which sublist to run through first, thus reducing the length of the search.

Another arrangement is to chain the production rules to one another, so that the consequent of one becomes the antecedent of another. For example, *if it is clear, then it is hot; if it is hot, then it is muggy; if it is muggy, then it is summer.* A number of these chains may be arranged in a tree structure or a graph of some sort. The search strategy then becomes a matter of searching this structure.

If nothing is known about which is the better path to follow, then the search may be either breadth-first or depth-first. In breadth-first, the search passes from node to node across the breadth of the tree structure. In depth-first, the search passes vertically down one branch of the inverted tree, then returns to the top and searches down the next branch, etc. Moreover, the domain expert may have some rules of thumb, or heuristics, that enable him or her to indicate that one part of the tree is more promising than the other parts. These heuristics may be incorporated in the search strategy.

The sample chain given above—*if it is clear, then it is hot; if it is hot, then it is muggy; if it is muggy, then it is summer*—leads to another complication. As phrased, the sequence indicates 100 percent certainty from one rule to the next. In reality, the probability that one rule follows the preceding one is much less than 100 percent in this case, so expert-system builders have devised various methods of incorporating uncertainty into their rules. For instance, the truth value of each production rule in this sequence may be only 0.5 on a scale of 0 to 1. The probability of all three rules being valid in a particular instance is still less.

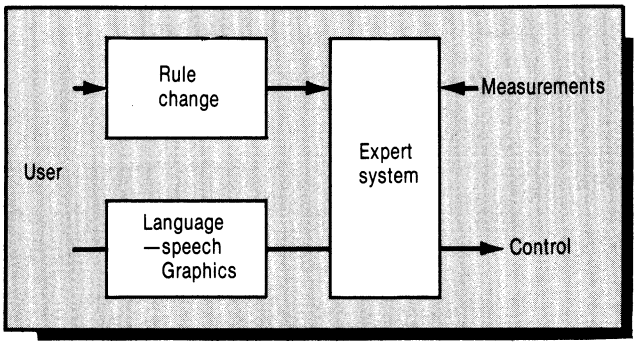


Figure 3. The expert system of the future will help a human operator control some machine, such as an aircraft.

Explanation trace. When the expert system comes up with the answer, *it is summer,* we may have some doubts about the correctness of this conclusion. The credibility of the system is greatly enhanced if it can explain to the user the line of reasoning that led to this finding. It can do this, essentially, by retracing the chain of production rules that led to this finding and translating them into some form readily intelligible to the user, such as English. Then with the user's common sense knowledge about, in this case, the local climate, he may check the expert system's chain of reasoning, noting the uncertainties associated with each step.

Expert assistant. A typical expert system supplies solutions in the form of data or information, and the user is responsible for making use of it. The concept of the expert system may be extended to the expert assistant, diagrammed in Figure 3. The expert assistant receives data directly from the environment in which it operates in the form of real-time measurements from sensors. With some direction from the user, the expert assistant generates signals in real time to control some aspect of the environment. The user interfaces to the expert assistant through a natural-language interface, a graphics interface, and ultimately speech. The user may change the rules under which the expert assistant is operating to help the assistant adapt better to the environment as it changes.

The expert assistant takes on more meaning if we think of it as a pilot's associate. An aircraft's instrument system now supplies the pilot with hundreds of readings, far too many to comprehend fully in an emergency. The pilot's associate could monitor this data, note interrelationships, figure out the significance of the information, and present a conclusion to the pilot in a form ready for implementation.

One step further, the pilot could preauthorize the pilot's associate to institute certain actions itself, particularly control actions where prompt response is critical. For this purpose the pilot's associate is connected directly to the aircraft's control system and itself takes the recommended action.

The rule-change capability enables the pilot to specify from time to time the authority granted to the system. For example, when the pilot wishes to practice certain flying skills, he removes that area temporarily from the jurisdiction of the pilot's associate, while leaving in force safety precautions. On the other hand, when he decides to devote his attention to some nonflying task, such as checking his position, he assigns more responsibility for operating the plane to the pilot's associate.

Instead of an aircraft, any kind of complex machinery or process could be directed by a human operator with an expert assistant. This complex machinery could be a robot.

AI languages and their hardware

The computer age got under way computing numbers, but artificial intelligence called for processing symbolic information, such as words and phrases. Aware of this need, John McCarthy invented Lisp in 1959. In 1972, Alain Colmerauer at the University of Marseilles began the development of Prolog, based on a different principle.

Lisp. Four commands are central to Lisp's symbol-manipulation capability. (The names of three of these commands are nonmnemonic as the result of "a regrettable historical convention," in Winston's phrase,[5] so we won't bother with them here.) One returns the first element in a list of elements, for example, the first word in a list of words. The second removes the first element from a list of elements and returns the remainder of the list. By using these two commands alternately, the second, third, or *n*th word in a list may be picked out. To put it another way, these commands take a list apart.

The next two commands put things together. One strings together the elements of two or more lists supplied as arguments. The other inserts a new first element in a list.

In addition, there are other commands that when applied to a predicate determine whether it is true or false. By definition a predicate is a function whose value is limited to true or false.

The last command that we will mention (there are many more) cruises down a list of "clauses," each composed of "a test and a result." When its argument matches the test side of a clause, it returns the corresponding result. In effect, the test and the result could be the two sides of an if-then rule. This capability may also be used to direct the flow of program control.

Thus, these basic commands can sort out symbols, can build up lists, can determine the truth or falsity of a function, and can match the if-side of a production rule. In general, Lisp's goal is to evaluate something and return a value.

Lisp has many advantages, and we will mention two: storage space is allocated dynamically, enabling programs to be larger than they would otherwise be; and there is a macro capability, permitting the language to be extended. In fact, Micro-planner, Conniver, Scheme, and other AI languages were written as extensions in Lisp.

Partly because of this extension capability, partly because of the need to adapt the language to different computers, and partly because of the tendency of researchers to implement new ideas, many versions of Lisp have been developed. These versions seem to have flowed primarily in two streams, one originating in the MIT Artificial Intelligence Laboratory, called Maclisp, and the other from Bolt, Beranek, and Newman, and later from Xerox Palo Alto Research Center, called Interlisp. Many of the Maclisp descendants are now in the process of being standardized as Common Lisp.

Prolog. Prolog, for programming in logic, was developed in Europe between 1970 and 1980. Alain Colmerauer, Robert Kowalski, and Phillip Roussel are the names associated with the early development. A compiler/interpreter was developed by David Warren, Fernando Pereira, and Lawrence Byrd at the University of Edinburgh in 1975. A modular version, MProlog, was produced in Hungary between 1979 and 1982.

The language received impetus from its selection in 1981 as the basis for the Japanese Fifth Generation Computer project. It is now gaining acceptance in the United States as well. For example, it is the language basis for the Aquarius project at the University of California at Berkeley.[6] This project is developing a high-performance computing system for combined symbolic and numeric applications. Also, Prolog is being used by organizations such as Applicon, Argonne National Laboratory, Caltech Jet Propulsion Laboratory, Fairchild, Gould SEL, Lockheed Missiles and Space Company, and SRI International.

The language works with English-like statements of three types: facts, rules, and questions.

(1) A fact is something such as *Richard works for Jean.* This fact indicates there is a relationship between Richard and Jean and that the relationship, as expressed, goes from Richard to Jean. This fact translates into Prolog like this:

works(richard,jean)

(2) A rule follows the form: Someone is the manager of an employee if the employee works for that someone. A rule is expressed as follows:

manager(X,Y) :- works(Y,X), where the expression ":-" is read as "if."

(3) A question might be: "Who is the manager of Richard?" It is expressed in this way:

?- manager(X,richard)

The Prolog language returns the answer:

X = jean

As we saw, a Lisp program consists of a series of commands that manipulate symbols. A Prolog program, on the other hand, consists of statements of facts and rules. This information is used by asking questions about it.

Prolog's adherents believe that it is easier to learn and use than Lisp. They say that it uses less memory and is more easily moved from one computer to another. In the past, it has run with reasonable speed only on mainframes, but recent modifications are running satisfactorily on smaller machines.

Software development tools for large computers

Advisor is from the ICL Knowledge Engineering Business Centre in Manchester, England. It runs on ICL VME mainframes and will be available on smaller systems. Written in Pascal, it costs about 15,000 pounds.

ART, Automated Reasoning Tool, is provided by Inference Corporation, Los Angeles. It runs on the machines from Lisp Machine and Symbolics and on the DEC Vax series. The price range is $60,000 to $80,000. It is a second-generation tool, the company says, distinguished from first-generation tools such as Emycin and OPS5 because it provides a sound base for commercial development, rather than exploring a paradigm for research purposes.

HPRL, Heuristic Programming and Representation Language, is being developed in the Computer Research Laboratory, Hewlett Packard, Palo Alto, California. It is considered to be an extension of FRL, Frame Representation Language, and to have goals in common with RLL and Age. It runs on the Vax 11/780 and the HP-9836.

IKE, Integrated Knowledge Environment, was announced by Lisp Machine at the 1985 International Joint Conference on Artificial Intelligence. It was introduced at $15,000 and runs on the company's Lisp machines. The package uses menu-based natural-language commands to customize the program's vocabulary to the application, to build the knowledge base, and to run interactive sessions to test the developing expert system. A graphic interface displays parse trees of rules as they are entered, thus verifying that the system has "understood" them.

KEE, Knowledge Engineering Environment, is available from IntelliCorp, Menlo Park, California. It runs on the Xerox 1100 series, Symbolics 3600, and the Lisp Machine, Inc., Lambda and is being made available on additional machines. The price is $60,000 for the first unit with substantial discounts for additional units. Release 2.0 in August 1984 extends the tool to support larger expert systems and adds a logic-based query language for KEE knowledge bases.

KES, Knowledge Engineering System, is supplied by Software A & E, Inc., Arlington, Virginia. It runs under various versions of Lisp, including Franz Lisp, Zetalisp, and Spice Lisp, a dialect of Common Lisp that runs on the Perq computer. In 1985, the tool was being reengineered in C to achieve further transportability, better performance, and the support of embedded applications.

Knowledge Craft is a product of the Carnegie Group, Pittsburgh, Pennsylvania. Priced at $50,000, including training and support, it runs on Texas Instruments' Explorer, the Symbolics 3600, and the DEC Vax series. It was derived from research at Carnegie-Mellon University, and the company considers it to be a "next generation" tool.

Picon, Process Intelligent Control, is the first of a series of tools being developed by Lisp Machine to make the technology more readily usable in particular industries. It helps operators deal with emergency situations in complex processes, such as refineries and chemical plants. The basic system may be adapted to different plants. It runs on the company's Lambda/Plus, meaning one Lisp machine plus one 68010 microprocessor. The purpose is to combine the real-time data-acquisition and numeric-processing capabilities of the microprocessor with the expert-system capabilities of the Lisp machine.

Reveal is from the ICL Knowledge Engineering Business Centre in Manchester, England. It runs on ICL VME mainframes and will be available on smaller systems. Written in Fortran, it costs about 30,000 pounds.

S.1 from Teknowledge, Inc., Palo Alto, California, is in the $50,000 class (less for additional copies). Written in Lisp, it runs on the Xerox 1108 workstation under Interlisp-D, Symbolics 3600 and 3670 machines in Zetalisp, and DEC Vax 11/750 and 11/780 under VMS in Franz Lisp. It is intended for large-scale systems, in contrast to the company's microcomputer tool, M.1.

S.1 Version 2 has been completely rewritten in the C language for use on computers that run the Unix operating system, including the AT&T PC7300, DEC Microvax, Xerox 1109, Symbolics 3670, NCR Tower 32, Gould Power Node 6031, Tektronix 4404, Sun-2/120, and Apollo DN30 and DN550. The development version, available in early 1986, will sell in the $25,000 to $40,000 range.

TIMM, The Intelligent Machine Model, from General Research Corporation, McLean, Virginia, is available in mainframe and personal-computer versions. Written in Fortran, the mainframe version runs on 4MB virtual machines with a full ANSI Fortran-77 compiler, including IBM, Amdahl, DEC Vax, and Prime. The first license is $39,500, less in quantity.

Lisp hardware. The Lisp language, of course, was developed on traditional, numerically oriented mainframes and later came to be widely used on minicomputers. Artificial-intelligence programs tend to be large and to use vast amounts of memory. Moreover, the structure of the language makes it execute slowly and inefficiently on this kind of computer. In addition, when several researchers had to timeshare a large computer, response time was still worse.

By the early 1970s, researchers realized that these problems could be ameliorated by developing an individual workstation optimized to run Lisp. The Lisp Machine project was launched at the MIT Artificial Intelligence Laboratory in 1974. A first-generation machine was completed in 1976 and a second-generation in 1978. Aware of a potential commercial market for the machine, two groups split off from the laboratory in 1980, one becoming Lisp Machine, Inc., with offices both in Andover, Massachusetts, and Los Angeles, and the other, Symbolics, Inc., in Cambridge, Massachusetts.

Also in 1974, researchers at Xerox PARC began an implementation of Interlisp for the Alto, a small personal computer internal to Xerox. The performance of this implementation, called AltoLisp, was not satisfactory for such reasons as the limited amount of main memory. By 1980, PARC had developed two more suitable personal computers, the Dorado and the Dolphin.

In general, the various Lisp machines took advantage of advancing technology to provide more main memory, more cache memory, stack architecture, faster processing speeds, pipelining and parallelism, and high-resolution interactive display screens. In addition, these machines implemented hardware specific features that directly supported the needs of Lisp.

In the descendants of the MIT development, the machine instruction set corresponds closely to Zetalisp, one of the successors to Maclisp. In the Symbolics 3600, for instance, a machine instruction takes the first element of a list and pushes it onto the stack.[7]

The 3600 features a 36-bit word and two of these bits are used to code the relationship between the first element and the rest of the list. This code avoids the need for using an additional word to hold a pointer to the next element, reducing memory needed for this purpose by one half.

In the operation of Lisp programs, some objects become orphans—they are no longer referenced by anything else and so they are no longer needed. Since memory space is valuable, a process called garbage collection identifies these objects and reclaims their space. Originally, this process had to read through the entire storage space to establish that there were no references to the stored objects being considered for disposal.

In the 3600, three hardware features speed up garbage collection: a two-bit field indicates whether the word contains a pointer, that is, a reference to a word in virtual memory (main memory plus disk); a page-tag bit indicates whether the garbage-collecting process need even scan the

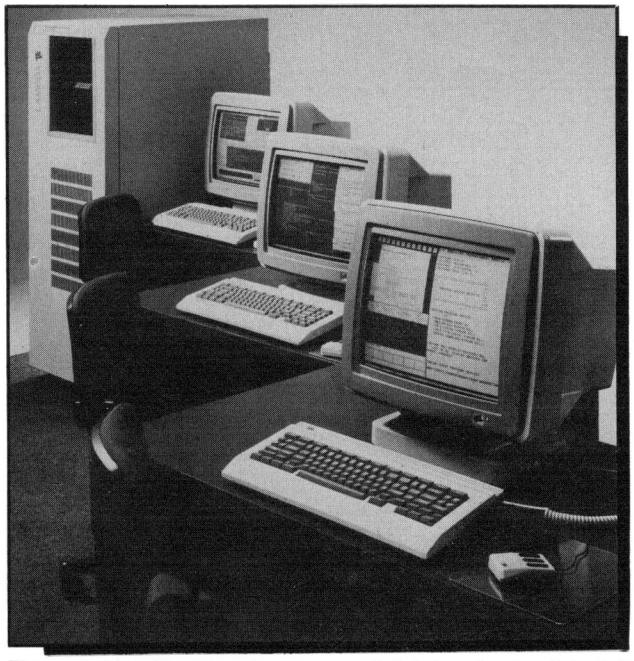


Figure 4. Lisp Machine, Inc., began delivering the Lambda 3x3 last summer (1985). The model designation refers to the fact that the system has three processors and accommodates three programmers. When an application requires great power, the $135,000 system's three Lisp processors may be pipelined together over a 37.5 MB/s multiprocessor bus.

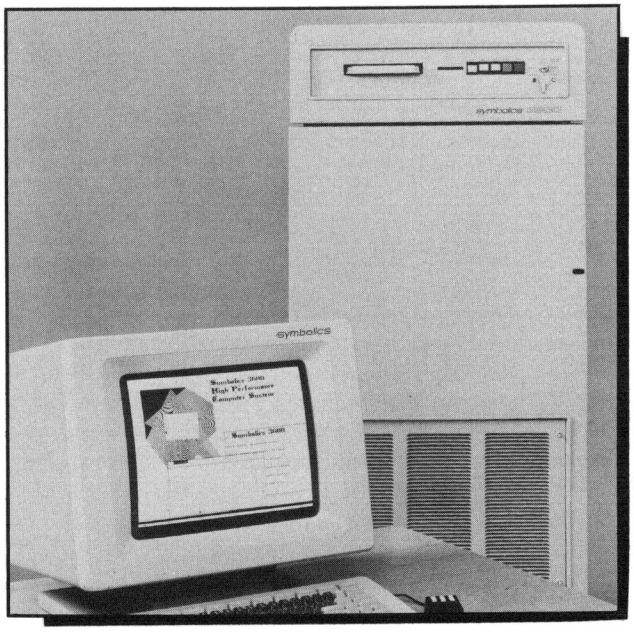


Figure 5. Introduced in 1983, the Symbolics 3600 supports the Zetalisp software environment and includes object-oriented programming or message passing. The entry-level configuration is about $80,000. Available as an option is the Interlisp Compatibility Package which accommodates Interlisp-10, Interlisp-VAX, and Interlisp-D.

Figure 6. In August 1984, Tektronix introduced the 4404 AI System and, in August 1985, the 4405 and 4406 AI Systems at about $12,000, $15,000, and $24,000. The three products feature the Smalltalk-80 programming environment. Tek Common Lisp, Franz Lisp, or MProlog are available as options.

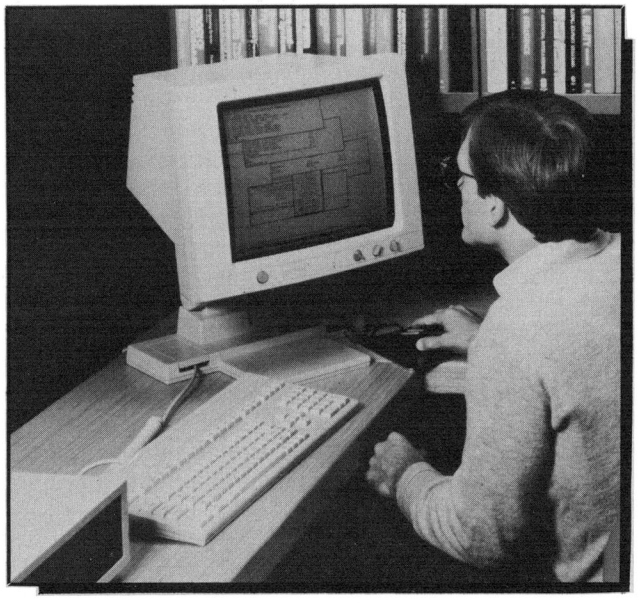


Figure 7. Texas Instruments' Explorer Symbolic Processing computer is derived from technology licensed from MIT and Lisp Machine. Introduced in late 1984, it is available in six configurations ranging from $52,500 to $66,500. The principal language is Common Lisp, but a Prolog interpreter may be called from the Lisp environment.

page; and multiword read-operations fetch several words at a time to the processor. In addition, garbage collection is incremental, running in the background.

Lisp Machine, Symbolics, (see Figures 4 and 5) and Xerox (in Pasadena, California) continue to produce workstations for artificial-intelligence applications. In addition, there are two recent entrants, Tektronix and Texas Instruments, illustrated in Figures 6 and 7.

At the low end of the machine spectrum are the personal computers. Versions of Lisp are beginning to appear for them. One is ExperLisp for the Macintosh from ExperTelligence, Santa Barbara, California, for $495. It is "the first complete implementation of Lisp on a microcomputer," the company says. According to an analysis by the president of the company, Denison Bollay, the relative performance of an 8088-based machine (IBM PC) is two, an 80286-based machine (IBM PC AT) or a 68010-based machine (Macintosh or Tektronix 4404) is 10, a future 68020-based machine would be 60, and the Symbolics 3600 is 100.[8]

As a foretaste of hardware to come, Texas Instruments is developing a custom VLSI Lisp processor chip based on sub two-micron CMOS technology for DARPA's Strategic Computing Program. The chip will be software-compatible with TI's Explorer Symbolic Processing workstation. "The chip is being designed to provide up to 10 times the processing power of today's commercial symbolic processors at substantially lower cost and physical size," the company says.

Prolog hardware. Thus far Prolog in different versions is running on general-purpose computers, including the DEC Systems 10 and 20, the DEC Vax series, and IBM's large mainframes, as well as the IBM PC and the Apple II. MProlog from Expert Systems, Ltd., Oxford, England, for example, is available on the IBM/370 under VM/370-CMS, the Siemens 7,000 under BS2000, and the Vax-11 under VMS. Quintus Computer Systems of Palo Alto, California, has tried to increase the availability of Prolog by developing portable, hardware- independent systems on standard general-purpose hardware.[9]

Machines aimed at running Prolog are under development. The Japanese Institute for New Generation Computer Technology, ICOT, is developing the PSI machine to maximize Prolog performance. NEC is working with ICOT on a higher performance machine, and ICOT projects a very high performance unit by 1992.

The Aquarius project at the University of California at Berkeley, under way since 1983, plans to radically improve performance by (1) using a variant of logic programming (initially Prolog) as the primary control mechanism for the problem-solving process, (2) tailoring the MIMD processing elements to the requirements of the intended set of ap-

plications, and (3) exploiting parallelism at several levels of concurrency.[6]

Again, as in the case of Lisp implementations, performance varies over a wide range. Kahn and Warren give the following figures:

Quintus Prolog on Sun-2 workstation under Unix: 20K LIPS (logical inferences per second);
Quintus Prolog on Vax ll under VMS and Unix 4.2: 23K LIPS;
The ICOT PSI machine (still under development): 33K LIPS;
The fastest Prolog on the market: 43K LIPS; and
Japanese goal for 1992: 100M LIPS[9]

Despain and Patt compare a dozen or more machines on several benchmarks.[6] Performance on general benchmark programs ranges from 10 LIPS (on the Apple II) to 205K LIPS. The latter figure is derived from a simulation of their Berkeley Programmed Logic Machine, the first experimental processor under the Aquarius project.

Software development tools

In the course of building the early expert systems, researchers noticed that the facts and if-then rules were specific to the problem domain, but the inference engines were very much the same from one domain to another similar one. Of course, in the first expert systems the knowledge bases and the inference engines were not neatly divided, but by the end of the 1970s researchers had sorted out a number of domain-independent versions of expert systems.

Emycin (for Essential Mycin) was derived from Mycin, a production rule-oriented expert system for diagnosing infectious blood diseases; KAS from Prospector, a minerals prospecting system; and Expert from Casnet, used in the diagnosis and treatment of glaucoma. These three are classified as skeletal systems.[3]

Representation languages are a class of general-purpose programming languages developed for knowledge engineering. Less constrained than skeletal systems, they may be applied to a broader range of tasks. This category includes Rosie, OPS5, RLL, and Hearsay-III.

A third category, computer-aided design tools for expert systems, is represented by the tool, Age. With it, the user may choose from several kinds of knowledge representation and processing methods.

With the domain-specific knowledge separated from the code in the inference engine, it would seem to be merely a matter of knowledge engineering to put together an expert system. Little programming should be necessary. Unfortunately, there may still be problems. The greatest one is that the inference tool fails to match the new problem area closely enough to be applied unchanged. A lesser one is that all task-specific knowledge has not been rooted out of the tool.

After investigating the use of these tools, Donald A. Waterman and Frederick Hayes-Roth offered some suggestions for choosing a tool appropriate to the problem:

"Do not pick a tool with more generality than you need.
Test your tool early by building a small prototype system.
Choose a tool that is maintained by the developer.
Choose a tool with explanation/interaction facilities when development speed is critical.
Use the problem characteristics to determine the tool features needed."[3]

The first large, comprehensive expert-system development tool to be marketed commercially, according to the company that developed it, was KEE, the Knowledge Engineering Environment, from IntelliCorp. It was introduced in August 1983 at the conference of the American Association for Artificial Intelligence. Since then, a dozen or more tools have appeared for use on mainframes and minicomputers. Still more recently, expert-system development tools to run on personal computers have become available.[10]

Shortcomings of tools. One tool vendor observes that most present-day tools do not support the integration of the expert system with existing software such as database management systems.[11] The ability to transport Lisp- or Prolog-based systems from one machine environment to another is limited. Their performance is often inadequate for production applications.

Silogic Incorporated of Los Angeles points out that current expert systems can operate only on expensive or special-purpose hardware and are too slow on small or conventional machines. Furthermore, they cannot operate on conventional machines in an industrial environment. They have problems going back and forth to a large database. They cannot use databases constructed on other systems.

In the process of interacting with its Fortune 100 clients, Teknowledge discovered that they were strongly wedded to their installed hardware and software base. "American and international business show no sign of abandoning their huge investment in currently installed hardware, software, databases, operating systems, and personnel training," according to Lee M. Hecht, the company's chief executive officer.

It was possible to persuade them to use Lisp-based or Prolog-based expert systems and the corresponding special-purpose computers at a research or development level. In general, they would not install hundreds of such systems for corporate-wide applications. There were logistic problems to introducing another type of hardware. Existing programming personnel were not comfortable with Lisp or Prolog.

Recognizing this reality, Teknowledge during 1985 reprogrammed its expert-system tools in C and ported them to conventional computers supporting the Unix operating environment. Moreover, in doing this, the company achieved "much higher performance," according to Earl D. Sacer-

doti, vice president and general manager of Knowledge Engineering Products and Training.

End of the beginning

Every one has seen glowing accounts of expert systems performing remarkable functions. It is true that some of the university-developed systems are now in practical application, that the university spin-off companies are developing useful systems, and that some large corporations have established artificial-intelligence groups.

"Today, there are fewer than 25 knowledge systems in the field," write Hayes-Roth and London, "but several hundred more are under development."[12]

As computer professionals, however, we should remember that, although the origins of artificial intelligence go back 30 years, the arrival of practical expert systems goes back little more than three years. As a new art, we should expect problems:

Tools for personal computers

ESP Advisor from Expert Systems International, King of Prussia, Pennsylvania, runs on MS-DOS 2.0 with 256K of memory. It is priced at $895.

ExperOPS5 was implemented by Science Applications International Corporation for ExperTelligence of Santa Barbara, California. The original OPS5 was created at Carnegie Mellon University. The present version, said by the company to be "a complete implementation," operates on a Macintosh 512K with ExperLisp and an add-on floppy or hard-disk drive. It was introduced in the summer of 1985 at a list price of $195.

Expert-Ease was developed by Jeffrey Perrone & Associates, Inc, San Francisco, and lists for about $700. It is written in Pascal and runs on IBM PCs and compatibles, DEC, and Victor. Perrone says that the tool enables users to produce "models" of processes that require specialized knowledge or skills. Models have been developed for property purchasing decisions, trouble shooting, analyzing logic designs, preliminary diagnoses of dental pain, assistance in building code compliance, analyzing expense claims, and recommending disciplinary actions. Perrone notes that some might not consider these to be "true" expert systems.

Exsys, from Exsys, Inc., Albuquerque, New Mexico, lists at $200 and accommodates up to 400 rules with 128K of memory or 3000 rules with 640K. The large version runs on MS DOS 2.00.

Insight 2 is a $495 product from Level 5 Research, Melbourne, Florida, that runs on IBM, DEC, and Victor personal computers with 192K of memory.

KDS, from KDS Corporation, Wilmette, Illinois, allows for up to 16,000 rules per knowledge module. The development system requires 256K, MS DOS 2.00, and is priced at $795.

Knowledge Workbench, based on Silog, a proprietary extension of Prolog, comes from Silogic Inc. of Los Angeles. The system incorporates not only a knowledge-base manager and inference engine, but a natural-language interface. It is being designed to run on MC68000-based personal computers.

M.1 and **M.1a** are Prolog versions of Teknowledge's microcomputer tool, the first at $10,000 and the second, $2000. Both operate under MS-DOS 2.0. M.1a is an evaluation subset of M.1 containing the basic features and sufficient power to develop a proof-of-concept system.

Last October Teknowledge announced the conversion of M.1 to C. The Version 2 development system is priced at $5000 with the delivery version at $500 per copy. Rule capacity has been expanded from 200 to 1000. Performance has improved four or five times over the original Prolog product.

MicroExpert at only $49.95 is a product of McGraw-Hill Book Co., New York. It requires 128K and MS DOS 2.00.

Micro KES, listing at $4000, is the personal-computer version of KES. It is implemented in IQLisp for the IBM PC XT with 640K. Software A & E says that it is functionally equivalent to KES running on a Vax, but its performance is much less—comparable to KES on a heavily loaded Vax. Thus, development on a personal computer is difficult, but execution is believed to be adequate.

NaturalLink Technology Package is a 1985 release from Texas Instruments at $1500 for TI's Professional Computer. It enables a programmer to construct English sentences that the underlying application program recognizes as commands. The ultimate user constructs an input sentence, not by typing, but by selecting words and phrases from lists of choices displayed in menus.

Personal Consultant is a Texas Instruments product that runs on TI's Professional Computer and is compatible with PC-DOS software products. Listing at $3000, it requires at least 512K. Using the Professional Computer, a user may develop an expert system of up to 400 rules. On a larger, higher capacity computer, such as TI's Explorer, he can go up to more than 1000 production rules and then run the developed system on the Professional Computer.

RuleMaster is available from Radian Corporation, Austin, Texas, in the $5000 to $15,000 range. It runs under MS-DOS 3.0 on the IBM PC XT or under Xenix on the IBM PC AT, and requires 600K.

TIMM-PC, The Intelligent Machine Model, is supplied by General Research Corporation, McLean, Virginia, at $9500 for the first license, less in quantity. It runs on MS-DOS 2.0 with 640K.

(1) There are two major languages, Lisp and Prolog, each supported with almost religious intensity by passionate advocates. Moreover, each exists in many versions. It is not clear which one or which version is best in some absolute sense, but it is evident that developers are successfully building systems in both languages and many versions.

(2) Others believe that conventional languages capable of running on conventional computers better meet the desires of the marketplace.

(3) There are more than a score of software development tools. They are often aimed at different applications; they may be based on different principles; they are implemented in different languages; and they operate on different computers. It takes considerable effort to find out which one is best for a particular application.

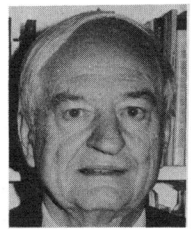
(4) There are mainframes and minicomputers, special-purpose Lisp machines and forthcoming Prolog machines, and workstations and personal computers. Which sizes and types are suitable for which applications?

(5) Moreover, there may be a great gap between a prototype and a successful product, experienced developers know. Some of the difficulties include: making the product more rugged, reliable, and user-friendly; arranging to run the software on hardware suitable for the application—hardware that is accessible and cost effective; defining a specific range of problems over which the system is to work, so that the user can easily recognize whether his particular problem is within the range served by the expert system; and meeting the users' need for an application they can afford.

(6) Some knowledgeable people believe that knowledge engineering and expert-system building are too complicated to be entrusted to engineers, programmers, or domain experts who are relatively inexperienced in the art. Others believe that the software development tools are sufficiently comprehensive to be used by people with little experience. Which belief is true now? How much further development of tools is necessary before inexperienced people will be able to build systems? Alternatively, what training and experience must people have to build expert systems? E

References

1. Robert M. Glorioso and Fernando C. Colon Osorio, *Engineering Intelligent Systems: Concepts, Theory, and Applications,* Digital Press, Bedford, Mass., 1980, 472 pp.
2. Pamela McCorduck, *Machines Who Think,* W. H. Freeman and Company, San Francisco, 1979, p. 96.
3. Frederick Hayes-Roth, Donald A. Waterman, and Douglas B. Lenat, editors, *Building Expert Systems,* Addison-Wesley Publishing Co., Reading, Mass., 1983, 444 pp.
4. Dana S. Nau, "Expert Computer Systems," *Computer,* Vol. 16, No. 2, Feb. 1983, pp. 63-85.
5. Patrick H. Winston, *Artificial Intelligence,* Addison-Wesley Publishing Co., Reading, Mass., 1977, 444 pp.
6. Alvin M. Despain and Yale N. Patt, "Aquarius—A High Performance Computing System for Symbolic/Numeric Applications," *Digest of Papers,* Compcon Spring 85, pp. 376-382.
7. Curtis B. Roads, Symbolics 3600 Technical Summary, Symbolics, Inc., Cambridge, Mass., 1983, 140 pp.
8. Tom Manuel, "The Pell-Mell Rush Into Expert Systems Forces Integration Issue," *Electronics,* July 1985, pp. 54-59.
9. Patti Kahn and David Warren, "Application Development in Prolog," *Proc. Artificial Intelligence & Advanced Computer Technology Conf. 1985,* Tower Conference Management Co., Wheaton, Ill., pp. 207-212.
10. Tom Schwartz, "AI Development on the PC: A Review of Expert System Tools," *The Spang Robinson Report,* Nov. 1985, pp. 7-14.
11. Andrew B. Ferrentino, "Expert System Development Environments: Current Limitations and Future Expectations," *Proc. Artificial Intelligence & Advanced Computer Technology Conf. 1985,* Tower Conference Management Co., Wheaton, Ill., p. 195.
12. Frederick Hayes-Roth and Phillip London, "Software Tool Speeds Expert Systems," *Systems & Software,* Aug. 1985, pp. 71-75.



Ware Myers is a freelance writer specializing in computer subject matter and a contributing editor for *IEEE Expert*. From 1965 until Xerox withdrew from the mainframe business in 1975, he was a member of the Systems Development Group, Computer Systems Divison, in El Segundo, California, where he worked on the development of analog instruments, color display stations, a microprogrammed controller, and several MOS memories. His principal contribution to these developments was the preparation of design specifications, technical descriptions, operating instructions, and reference manuals. From 1956 to 1965 he was with the Consolidated Electrodynamics Corporation and its subsidiary, Consolidated Systems Corporation. He has also worked as an instructor and lecturer in engineering at the University of California, Los Angeles.

Myers received his BS from the Case Institute of Technology and his MS from the University of Southern California. He is a member of Tau Beta Pi and the IEEE Computer Society.

Myers' address is c/o *IEEE Expert*, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720.