EXPERT SYSTEMS

You and your family just found the perfect house, and now all you have to do is get the XYZ Mortgage Corporation to approve the loan. You go to your neighborhood branch and talk to the loan officer. After filling out multiple forms and recounting your life history, the loan officer says, "You are in luck; the loan committee is meeting tomorrow morning, and they should be able to make a decision on your loan approval status by tomorrow afternoon." You say great, and off you go. Have you ever asked yourself the question, "Who is on the loan committee?" Well, the answer, in today's modern technology world, may be your friendly personal computer.

That's right; a computer may be deciding whether your loan is approved or denied. In many fields of business, the sciences, and government, computers, programmed with the decision-making expertise and knowledge of a human, are actually making everyday decisions. As business and government strive to cut costs and be more productive, many decisions are being made by computers rather than humans, using expert systems.

This article addresses the technology known as expert/knowledge-based systems: their definition, history, structure, development, and their future status. This article is meant to serve as an introduction to the field of expert/knowledge-based systems and the many problems, both big and small, that can be solved using this important computing technology.

Before continuing, we need to clarify the terminology used in this article. While this article addresses expert systems, the term is often inaccurately used. Expert systems actually refer to systems that exclusively use human expertise to solve decision-making problems; however, a broader class of technology is often referred to when discussing expert systems. Expert systems are a subset of knowledge-based systems, which are a class of decision-making computer technology that uses domain-specific knowledge, from possibly many sources, to solve critical problems. Therefore, it is more accurate to refer to the technology in this article as knowledge-based systems. However, we will use the term *expert systems* in keeping with the main intent of this article and for the sake of clarity.

Specifically, this article is divided into five major sections. In the remainder of this section, we first give a definition of an expert system. Next, we briefly discuss the historical aspects of this technology, including its relation to the broader field of artificial intelligence and some of the significant expert systems that have been developed. We then review the major application areas for these systems and highlight some significant books, journals, and conferences that feature the discussion of the application of expert systems to real-world problems.

In the second section, we focus on the structure and major components of an expert system. The structure of an expert system differs from conventional procedural programming (e.g., programs written in C) in that the data (knowledge) in the system resides in a knowledge base and its distinct and deliveratly separated from the control mechanisms that reside in the inference engine. In the third section, we discuss the process of development of these systems. One of the distinguishing features of expert system development is that they are primarily built using a rapid prototyping paradigm (1). The fourth section reviews some of the most current applications and corporate usage of expert systems. These applications, both large and small, show the variety of application domains that are being addressed by these systems. We then look into the future for expert systems technology. We discuss some of the key research areas that need to be addressed in order to make this technology more applicable, and we describe the evolution to the next generation of expert system technology.

EXPERT SYSTEMS: A DEFINITION

The primary intent of expert system technology is to realize the integration of human reasoning into computer processes. This integration not only helps to preserve the human expertise but also allows humans to be freed from performing the mundane activities that might be more readily assigned to a computer-based system.

Given the number of textbooks, journal articles, and conference publications about expert systems and their application, it is not surprising that there exist a number of different definitions for an expert system. In this article, we use the following definition:

> An expert system is an analog to human reasoning in a clearly defined domain of expertise. Given a set of critical information in the form of facts, it can draw a conclusion similar to what one would expect from a human expert(s).

J. Webster (ed.), *Wiley Encyclopedia of Electrical and Electronics Engineering.* Copyright © 1999 John Wiley & Sons, Inc.

In order to fully understand and appreciate the meaning and nature of this definition, we highlight and detail the four major component pieces.

- An expert system is a computer program. A computer program is a piece of software written by a programmer as a solution to some particular problem or client need. Because expert systems are software products, they inherit all the problems associated with any piece of computer software. Some of these issues will be addressed in the discussion on the development of these systems.
- An expert system is designed to work at the same (or higher) level of decision-making ability. The specific task of an expert system is to be an alternative source of decision-making ability for organizations to use, instead of relying on the expertise of just one—or a handful—of people qualified to make a particular decision. An expert system attempts to capture the expertise of a particular person for a specific problem. Usually, expert systems are designed and developed to capture the scarce but critical decision-making that occurs in many organizations. Expert systems are often feared to be replacements for decision-makers; however, in many organizations, these systems are used to free up the decision-maker to address more complex and important issues facing the organization.
- An expert system uses a decision-maker(s) [i.e., expert(s)]. Webster's dictionary (4) defines an expert as

  > One with the special skill or mastery of a particular subject

  The focal point in the development of an expert system is to acquire and represent the knowledge and experience of a person(s) who have been identified as possessing the special skill or mastery.
- An expert system is created to solve problems in a clearly defined domain of expertise. The above definition restricts the term expert to a particular subject. Some of the most successful development efforts of expert systems have been in domains that are well scoped and have clear boundaries. Specific problem characteristics that lead to successful expert systems are discussed as part of the development process.

Now that we have defined what an expert system is, we will briefly discuss the history of these systems. In this discussion, we include their historical place within the artificial intelligence area and highlight some of the early, significant expert system development.

HISTORY OF EXPERT SYSTEMS

Expert systems are one of the two major paradigms for developing intelligent systems within the field of artificial intelligence. Expert systems are an example of the symbolic paradigm; the other major paradigm is the numeric paradigm that has led to the development of neural network technology. In order to discuss the history of these systems, a brief history of the artificial intelligence field is necessary. Expert systems were the first major successful application technology to evolve from artificial intelligence research.

Artificial Intelligence

The foundations of the field of artificial intelligence can be traced from many different disciplines including philosophy, mathematics, psychology, computer engineering, and linguistics (5).

The first cited work in the area of artificial intelligence dates back to McCulloch and Pitts (6) in 1943. They proposed a model of artificial neurons that mimic the structure of the human brain.

In the summer of 1956, John McCarthy organized a two-month workshop at Dartmouth, and 10 leading U.S. researchers interested in automata theory, neural networks, and the study of intelligence were invited (5). Two researchers from Carnegie Tech (now known as Carnegie Mellon University), Allen Newell and Herbert Simon, were the focus of the workshop due to their reasoning program known as the Logic Theorist (LT). Simon claimed, "We have invented a computer program capable of thinking non-numerically, and thereby solved the venerable mind-body problem." Soon after the workshop, LT was able to prove most the theorems in Chapter 2 of Russell and Whitehead's *Principia Mathematica.* An interesting note is that a paper on the use of LT to prove the theorems was rejected by *The Journal of Symbolic Logic.* The Dartmouth workshop accomplished two major outcomes. First, it served as a forum to introduce the leading researchers to each other; for the next twenty years, the field of AI would be dominated by these ten individuals, their students, and colleagues at MIT, CMU, Stanford, and IBM. The second major accomplishment of the workshop—and a more lasting one—was an agreement to adopt John McCarthy's new name for the field: Artificial Intelligence (AI).

The work of Newell and Simon is the first documented work using the symbolic programming paradigm of AI. Their work on LT led them to develop another program known as General Problem Solver (GPS). The success of GPS was not as widely heralded, however, because of the limited class of problems that it could solve. GPS was designed from the start to imitate human problem-solving protocols regardless of the information contained in the domain. These so-called "weak" methods—because they use weak information about the domain—turned out to show weak performance in solving problems in more complex domains.

Another significant event that helped propel expert system development was the definition of a high level programming language known as LISP (LISt Processor). LISP was developed by John McCarthy in 1958 to help develop symbolic-based computer programs. LISP, the second oldest programming language, later became the dominant AI programming language.

Since weak methods of problem-solving proved lacking in performance, other researchers took the opposite approach in the development of the DENDRAL program (7). They applied the knowledge of analytical chemists to infer the molecular structure from the information provided by a mass spectrometer. DENDRAL holds a significant place in the history of expert systems because it was the first system to use the expertise of human problem-solvers and translate that knowledge into a large number of special purpose rules, known as a rule-based system.

Early, Significant Expert Systems

The work on DENDRAL led to many others successful applications of this new technology known as expert systems.

Feigenbaum and others at Stanford began the Heuristic Programming Project (HPP) to investigate other problem domains that could benefit from this new technology. The next major effort was in the area of medical diagnosis. Bruce Buchanan and Dr. Edward Shortliffe developed MYCIN to diagnose blood infections (8,9). Using about 450 rules, MYCIN was able to perform as well as some experts, and considerably better than some junior doctors were.

MYCIN is one of the most widely known of all expert system applications. And this is despite the fact that it has never been put into practice. However, MYCIN is significant to the history of expert systems for two particular reasons. First, unlike DENDRAL, which used a model of a particular molecule as the basis for its reasoning, MYCIN was constructed from interviews with various doctors in the particular domain. Therefore, MYCIN contains a number of heuristic rules that are used by physicians in the identification of certain infections. The second major contribution of MYCIN was the later development of EMYCIN (Empty MYCIN). EMYCIN was the first expert system shell. It took approximately 20 man-years to develop the MYCIN program. The researchers realized that if expert systems were to become a viable problem solving technique, this development time must be cut. In an effort to reduce the time to develop an expert system, the researchers developed EMYCIN by taking all of the rules out of the system and leaving just an empty shell in which other developers in other domains could then plug in their new knowledge base. We discuss expert systems shells in the development section.

There were other significant expert system applications that were also developed in the early days of expert systems. These systems include PUFF, which used EMYCIN in the domain of pulmonary disorders, and DELTA/CATS, which was developed at General Electric Company to assist railroad personnel in the maintenance of GE's diesel-electric locomotives (10). Also at this time, researchers at CMU developed the first truly successful commercial application of expert systems. The system, developed for Digital Equipment Corporation (DEC), was used for computer configuration and known as XCON (R1).

XCON, originally titled R1, was developed by John McDermott at CMU for aiding in the configuration of VAX and PDP-11 computer systems at DEC. There exists an enormous number of configurations for VAX and PDP-11 computer systems; DEC attempts to configure each system to meet specific customer needs. XCON was originally developed as a 500-rule prototype that examined the specific needs of the customer and decided the exact configuration of components necessary to meet the customer requirements. In particular, XCON's function was to select and arrange the components of a computer systems including the CPU, the memory, the terminals, the tape and disk drives, and any other peripherals attached to the system. XCON works with a large database of computer components, and its rules determine what makes a complete order.

The development effort began in 1978, and by September 1979, XCON was able to configure more than 75 percent of all customer orders that it was given. By 1981, XCON was being used by DEC on a regular basis, and DEC estimates that its cost savings in 1983, 1984, and 1985 were a combined $83 million. Today, XCON is still being used by DEC to configure all VAX orders. There is a development team dedicated to keeping the rules in XCON current and keeping the users of XCON trained on the latest updates. A new copy of XCON is released practically every 3 months, and the latest version handles nearly 12,000 different computer components that could possibly configured into a customer order (11). XCON is one of the major, early success stories in the field of expert systems, for its high visibility domain, its continued use and expansion, and its tremendous impact on the bottom line (profit) at DEC.

MAJOR APPLICATION AREAS

There are two different ways developers look at application areas for expert systems. First, they look at the functional nature of the problem. Second, they look at the application domain. We review both of these ways to get a better understanding for the application of expert systems to "real-world" problems. In 1993, John Durkin (12) published a catalog of expert system applications that briefly reviews a number of applications of expert system technology and categorizes each of the nearly 2,500 systems.

Both MYCIN and XCON point out two different functions that are viewed as highly favorable for expert system development. MYCIN mainly deals with the diagnosis of a disease given a set of symptoms and patient information. XCON, on the other hand, is a synthesis-based (design) configuration expert system. It takes as its input the needs of the customer and builds a feasible arrangement of components to meet the needs. Both of these systems solve different generic types of problems.

An expert system may have many differing functions. It may monitor, detect faults, isolate faults, control, give advice, document, assist, etc. The range of applications for expert system technology ranges from highly embedded turnkey expert systems for controlling certain functions in a car or in a home to systems that provide financial, medical, or navigation advice to systems that control spacecraft.

Table 1 lists the ten different types of problems generally solved by expert/knowledge-based systems. Within each problem type, experts perform a generic set of tasks, such as diagnosis or planning.

As can be seen from Table 1, there are many different types of problems that can be solved using expert system technology. Currently, the majority of expert system applications are diagnostic systems [Durkin (12) estimates nearly

Table 1. Heuristic Problem Classification of Expert Systems Application Areas

Problem Type	Description
Control	Governing system behavior to meet specifications
Design	Configuring Objects under constraint
Diagnosis	Inferring System Malfunction from observables
Instruction	Diagnosing, debugging, and repairing student behavior
Interpretation	Inferring situation description from data
Monitoring	Comparing observations to expectations
Planning	Designing actions
Prediction	Inferring likely consequences of given situation
Prescription	Recommending solution to system malfunction
Selection	Identifying best choice from a list of possibilities

30%]; interpretation and prediction systems are also highly favorable functional domains.

Expert systems also cover a number of different application areas, such as business, manufacturing, medicine, and engineering. Durkin lists over 20 different application areas, including business, which encompasses marketing, management, finance, accounting, and so on.

BOOKS/JOURNALS/CONFERENCES

Many books, articles, and conference proceedings have been published over the years discussing the design, development, testing, and application of expert systems technology. Our purpose here is not to categorize all of this tremendous literature but to highlight some of the authoritative works in the field. One of the first textbooks on expert systems to appear was published in 1986 by the late Donald Waterman entitled *A Guide to Expert Systems* (13). At the same time, a number of textbooks and edited volumes dedicated to describing the development methods for expert systems and their various applications began to appear (9,14,15). More recent textbooks on expert systems have been written (10,16,17,18). Each of these textbooks provides a solid introduction to the development and application of expert systems. Another source of introductory information into expert systems can be found in chapters contained in many artificial intelligence textbooks (5,19,20).

Recently, the impact of periodicals (professional journals) on AI research has been examined (21). Many of these journals regularly feature development and application articles on expert systems technology.

There are a number of professional organizations are involved in promoting and discussing expert system technology, including American Association of Artificial Intelligence (AAAI), IEEE Computer Society, Association for Computing Machinery (ACM), and Decision Sciences Institute (DSI).

Many conferences are designed to act as a forum for discussion of expert systems including the biannual *World Congress on Expert Systems* and *Expert Systems,* which is sponsored by the British Computer Society.

To this point, we have provided an overview of expert systems by presenting a definition, reviewing the history and some successful applications, and recommending various starting points for research into the field of expert systems. In the next section, we will begin to examine the structure of an expert system and discuss, in some detail, the major components that make this technology unique.

STRUCTURE OF EXPERT SYSTEMS

In the early days, the phrase *expert system* was used to denote a system whose knowledge base and reasoning mechanisms were based on those of a human expert. In this article, a more general position is held. A system will be called an expert system based on its form alone and independent of its source of knowledge or reasoning capabilities.

The purpose of this section is to provide an intuitive overview of the architectural ideas associated with expert systems. In discussing the architecture of expert systems, we will first introduce the concept of an expert system kernel and then embed that kernel in a fuller and more traditional expert system architecture.

EXPERT SYSTEM KERNEL ARCHITECTURE

The kernel of an expert system contains those components that are the basic and required components for all expert systems. These components are identified as a fact base, a rule base, and an inference mechanism. The fact base and the rule base combine to become the knowledge base for the kernel.

Figure 1 provides an overview of the kernel of an expert system from this structuralist point-of-view. At the highest level of abstraction, there is the environment E and the expert system ES connected to or embedded in it. This level may be represented as

$$E \leftrightarrow ES$$

The environment establishes the domain of application of the expert system.

In addition to being viewed as the context in which the expert system performs its functions, the environment E may be viewed as the source of the knowledge that the expert system ES has and the data which drives its behaviors. The expert system ES may be viewed as a reactive system; that is, it reacts to data and information it receives from the environment E based on the reasoning capabilities it possesses.

Knowledge Base

In our discussion, we will consider only rule-based expert systems. In a rule-base expert system, the knowledge of the domain is captured (represented) by production rules (22).

The knowledge base in an expert system kernel consists of both a fact and a rule base. The fact base contains up-to-date (dynamic) information and data on the state of that portion of the environment E that is pertinent to the expert system kernel. The rule base is typically populated with rules (static) of the following form:

$$A \rightarrow B$$

This is interpreted as "if condition A is satisfied, then do B." The A portion of the rule is called the antecedent or LHS (Left Hand Side) of the rule. The B portion of the rule is called the consequent or RHS (Right Hand Side) of the rule. If A is true (i.e., all of its conditions are satisfied by data and facts

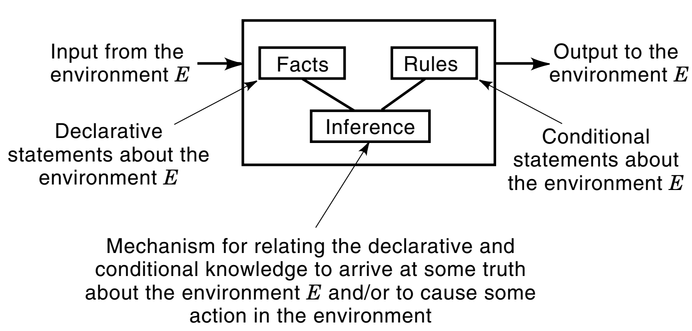

Figure 1. The kernel of an expert system contains the necessary components of the system.

in the fact base) and whatever actions specified in B are accomplished, then the rule is said to have been *fired.*

The condition A may be a conjunction of conditions $A_1, A_2, \ldots, A_n$, which must all be satisfied in order to trigger any actions stipulated by B. Any component of this conjunction may involve a negative. Likewise, B may be a sequence of actions $B_1, B_2, \ldots, B_k$, all of which will be taken if the conditional part of the rule is satisfied and the rule is fired.

The relationship between the rule base and the fact base is quite straightforward. If there is a fact in the fact base like "$\mathrm{Var}_1 = n$" and there is a rule in the rule base that states that "If $\mathrm{Var}_1 = n$ then B," then this rule is considered for execution (or firing) (known as triggering). There may be several rules that are candidates for firing based on the status of the fact base; this makes up the conflict set. It is up to the inference mechanism to resolve any conflicts and determine the appropriate rule to fire.

Inference Engine

The inference engine (mechanism) is that part of the expert system kernel which supports reasoning about the environment by proper manipulation of its rule and fact bases. It establishes the current state of the environment from its fact base and uses that state information to identify the set of rules whose conditional parts are satisfied by the environment's state. It determines which rules in the rule base are possible candidates for firing based on the circumstance that the conditional part of the rules are satisfied by facts in the fact base. These facts provide an up to date picture of the environment for the expert system.

There are basically two ways, or control strategies, by which the inference engine manages rules to arrive at some conclusion or to arrive at a sequence of actions to be taken with respect to the environment. These are forward and backward chaining. Most expert systems support only one control strategy. Some support both.

Forward Chaining. Forward chaining supports what is called data-driven reasoning. It is especially important for monitoring functions. Forward chaining works from LHS to RHS of rules.

$$A \to B$$

$$\longrightarrow$$

Forward Chaining

To get an intuitive feeling for this type of chaining, consider the following procedure:

- Identify new facts and data in the fact base
- Identify the rules whose LHSs are satisfied by the selected data and facts
- If more than one rule is identified, resolve conflict and select one rule or sequence of rules according to some priority
- Fire the rule or sequence of rules

The activation of the RHS of the selected rule(s) will result in new facts and data being instantiated in the fact base. These new data facts can again be used to identify rules whose LHS are satisfied and the forward chaining process can proceed. This process continues until no new facts are instantiated.

Backward Chaining. Backward chaining supports goal-driven reasoning. It is especially important for diagnostic activities. Backward chaining works from RHS to the LHS of rules:

$$A \to B$$

$$\longleftarrow$$

Backward Chaining

In this type of control strategy for managing rules, the initial focus is one the RHS of some selected rule from a set of rules whose RHSs satisfy some selected goal. The idea is to identify the conditions of the environment that would be necessary to achieve a selected goal. Consider the following for an intuitive feel for the process:

- Select goal to be achieved
- If goal is solved, return true
- Else,
  - Identify rules in the rule base whose RHSs reflect the goal
  - Examine the LHS of selected rules
  - Identify the facts and data in the fact base needed to satisfy the LHSs

Using the identified facts as new subgoals and going through the identified process, backward reasoning continues until a goal is proven true.

AN EXPERT SYSTEM ARCHITECTURE

If we embed the kernel of an expert system in an operational context—that contains processes for interacting with and interfacing with a user, a process for knowledge and data acquisition, and a process to support the generation of explanations for rule firings and advice to the user—then we arrive at what is customarily viewed as the architecture for an expert system.

Figure 2 displays the architecture commonly associated with expert systems. In our terminology, it is comprised of a kernel augmented by processes for data and knowledge capture, user interfaces and interactions, and a process for generating and presenting to a user explanations of its behaviors.

The "knowledge and data acquisition" process is used by the expert system to acquire new facts and rules associated with its specific domain. It is through this process that "knowledge" can be added to or subtracted from the expert system. Associated with this process is the concept of knowledge engineering. This is the process whereby knowledge from an expert or group of experts or other sources such as books, procedure manuals, training guides, etc. are gathered, formatted, verified, and validated, and input into the knowledge base of the expert system (see the discussion on expert/knowledge development for a more detailed explanation of knowledge engineering activities).

The "user interface" process is the mechanism used by the expert system to present to some human user information on

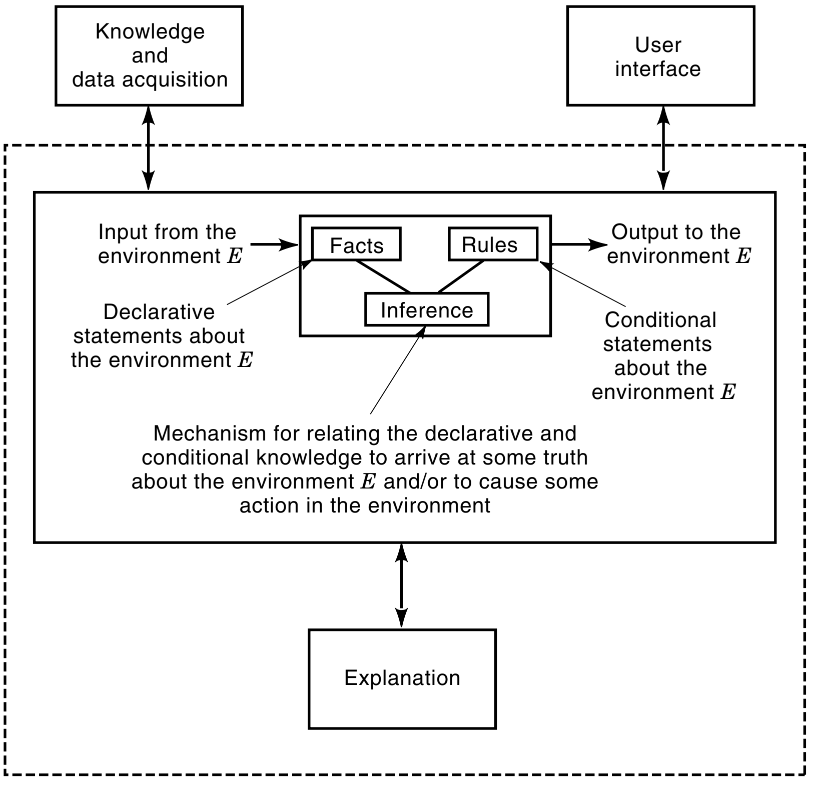

Figure 2. The expert system architecture contains the kernel of the expert system as well as the support tools for expert systems development.

its functioning, and specifically information on its determination of the state of the environment to which it is associated and its actions relevant to its understanding of the environment's state. Most current user interfaces are supported by multimedia technology and are designed to provide the user with the most complete and unambiguous presentation of information possible.

The "explanation" process is used by the expert system to provide to the user a trace of its actions and/or recommendations. This explanation is usually generated by providing a textual commentary identifying the sequence of rules it has fired with associated canned or automated commentary generation on why the rule was fired. This type of explanation can be used by the user to verify that the reasoning mechanism being utilized by the expert system is correct. It also provides additional information to the user that can be used to establish a more complete context for understanding both the state of the environment in question and the rationale for any advice or opinion given by the expert system.

DEVELOPMENT

The development of an expert system, often referred to as knowledge engineering, follows much the same path of any other software product. However, within the development of an expert system, the terminology and the nature of the software development process are different from conventional software systems.

The major development effort in creating an expert system is the design and development of the knowledge base (KB). One of the problems with the design and development of a KB is the lack of a *formal* methodology. By formal methodology, we mean a strategy that allows us to measure (precisely) the performance of an expert system similar to conventional software system design and development.

Expert system development (usually) relies on an evolutionary rapid prototyping methodology to create the KB. One definition of rapid prototyping is an iterative process that develops "an easily modifiable and extensible *working model* of a proposed system, not necessarily representative of a complete system, which provides users of the application with a physical representation of key parts of the system before implementation" (1). By using rapid prototyping, the developer can focus on building small working systems that can be the central element of discussions between the users, clients, and developers in solving the particular decision problems at hand. The rapid prototyping paradigm for KB development is often unstructured and ad hoc, especially concerning the testing and evaluation of the KB. This can lead to the development of a KB that is inefficient and contains numerous potential errors. Under evolutionary rapid prototyping, an expert system is designed and built in an incremental fashion.

There have been many paradigms offered for the design and development of an expert system. The best known of these paradigms is the five-stage process given by Buchanan et al. (23). These five stages—identification, conceptualization, formalization, implementation, and testing—correspond loosely to the eight stages in the waterfall model for conventional software development. Buchanan et al. points out that the process for developing an expert system developer and the domain expert may revisit any of the previous stages for further revision or refinement of the concepts and/or relationships in the problem domain. This is inherent in the evolutionary rapid prototyping process.

Derek Partridge (24) describes a methodology of artificial program construction through a process known as RUDE (Run-Understand-Debug-Edit). RUDE is based on rapid prototyping and the abstraction of the problem at each stage of the development process. He describes an expert system as an incompletely specified function because it models the behavior of a human and, as such, is not formally specified in the same manner as conventional software. Partridge argues that given this incomplete problem specification, the only way to develop an expert system is through a trial-and-error approach. Many other approaches have been proposed (13,14,25,26).

An approach that builds on both Buchanan and Patridge is the four-stage methodology known as DICE (3); DICE stands for Design, Implementation, Critique, and Editing. The methodology, which emphasizes testing and reliability analysis, uses evolutionary rapid prototyping and creates a control system where the feedback of the testing results improves the reliability and performance of the system.

Regardless of the methodology chosen to develop an E/KBS, there are six key activities to be performed within the development life cycle of an expert system:

- Problem selection
- Knowledge acquisition
- Knowledge representation
- Implementation
- Testing, verification, validation, evaluation
- Maintenance/sustenance

In this section, we discuss each of these activities in relation to the development of an expert system.

Problem Selection/Feasibility

Someone once said that there are three impotant rules in developing an expert system. The first rule is pick the right problem, the second rule is pick the right problem, the third rule is pick the right problem. In software development and scientific research, the most critical step is choosing the problem (27). Especially in the area of knowledge engineering, problem selection is critical. Finding a problem of the proper scope is especially important in expert system development. Remember that expert systems solve problems in a clearly defined domain. If the domain is too large, acquisition of the proper knowledge becomes an overwhelming task; if the domain is too small, the solution looks trivial. However, it is especially important to ensure full coverage of the entire domain.

In this section, we give guidelines for selection of the proper expert systems application problem. The majority of this discussion comes from work done by David Prerau from work done of COMPASS and other systems in the telecommunications domain (28). These problem selection guidelines are discussed in terms of the type of problem, the expert, and the domain area personnel.

The knowledge engineering team (the developers) should follow these guidelines for the selection of the appropriate problem:

- The task requires symbolic reasoning.
- The task requires the use of heuristics.
- The task may require decisions to be based upon incomplete or uncertain information.
- The task does not require knowledge from a large number of sources.
- Good sets of test cases are available.
- A few key individuals are in short supply.
- The domain is one where expertise is generally unavailable, scarce, or expensive.
- The task is decomposable, allowing rapid prototyping for a small, closed subset of the complete task and then slow expansion to the complete task.
- The task solves a problem that has value but is not on a critical path.
- The amount of knowledge that is required by the task is large enough to make the knowledge base developed interesting.
- The task is sufficiently narrow and self-contained. The aim is not to build a system that is expert in an entire domain, but a system that is expert in a limited task within the domain.
- The domain is characterized by the use of expert knowledge, judgment, and experience.
- Conventional programming (algorithmic) approaches to the task are not satisfactory.
- There are recognized experts that solve the problem currently.
- Expertise is not or will not be available on a reliable and continuing basis; that is, there is a need to capture the expertise.
- The system can be phased into use gradually. Incomplete coverage can be tolerated (at least initially), and it can be easily determined whether a subproblem is covered by the present system.
- The task is not all-or-nothing; some incorrect or nonoptimal results can be tolerated.
- The skill required by the task is taught to novices.
- Solution does not require the use of common sense.
- There are written materials that discuss the domain.
- Experts would agree on whether the system's results are good (correct).
- The need for the task is projected to continue for several years.
- Management is willing to commit the necessary human and material resources.
- The task requires only cognitive skills, not perceptive (vision, tactile, auditory, etc.) skills.
- The task should be performed frequently.

To summarize these guidelines, a good problem to solve is one that is cognitive in nature and sufficiently complex, has the support of management and users, and has been shown to be an important function provided by only one person (or a small group) frequently.

Another critical factor in the development of an expert system is having an expert to work with the knowledge engineering team. The following is a set of guidelines for what characteristics make a good expert:

- There is an expert who will work on the project.
- The expert's knowledge and reputation must be such that if the system captures a portion of the expertise, the system's output will have credibility and authority.
- The expert has built up expertise over a long period of task performance.
- The expert will commit a substantial amount of time to the development of the system.
- The expert is capable of communicating his or her knowledge, judgment, and experience, as well as the methods used to aply them to a particular task.
- The expert is cooperative.
- The expert is one person the company can least afford to do without.
- The expert should have a vested interest in obtaining a solution.
- The expert must also understand what the problem is and should have solved it quite often.

In summary, you would like to find a domain expert that is cooperative, articulate, and considered knowledgeable by others in the company.

The third major group involved in the development of an expert system is the domain area personnel (users and managers). As stated above, it is essential to have the support of the people for whom the system is being developed. These guidelines provide a set of criteria related to the domain area personnel during the problem selection phase:

- Personnel in the domain area are realistic, understanding of the potential uses and limitation of an expert system for their domain.
- Domain area personnel understand that even a successful expert system will likely be limited in scope and, like the human expert, may not produce optimal or correct results all the time.
- There is a strong managerial support from the domain area, especially regarding the large commitment of time by the expert(s) and their possible travel or temporary relocation, if required.
- The system developers and domain area personnel jointly agree upon the specific task within the domain.
- Managers in the domain area have previously identified the need to solve the problem.
- The project is strongly supported by a senior manager for protection and follow-up.
- Potential users would welcome the complete system.
- The system can be introduced with minimal disturbance of the current practice.
- The user group is cooperative and patient.

In summary, the domain area personnel have to be involved at every step of the development process. The users and managers should be shown intermediate prototypes, and feedback from their interaction should be included in subsequent prototypes. In addition, there is nothing in the above list of guidelines that cannot be stated for any software development project. By involving the users and managers in the process, you can significantly increase the chances of the final system being a product that is useful and potentially cost-effective for the organization.

Knowledge Acquisition

To get knowledge into a computer program, we must acquire it from some source. This section considers the manual acquisition of knowledge. Automated approaches are discussed in the section on learning.

Two major sources exist for the knowledge used in expert systems: experts (which could include the expert system developer) and documents or text. Both sources have advantages and disadvantages, although human experts are most always preferred. Experts tend to be more current and have a broader range of knowledge than documents. They also can respond to questions and provide different sets of examples. However, their time is expensive, and unless they support the project, they can work against the goals of the expert systems development. In some cases, expertise may have been lost, and the developer must rely on documents. Documents are generally cheaper to acquire and use. However, they typically have limited amounts of information, and what they have is not always completely relevant.

There are two major methodological components of knowledge acquisition from experts: acquisitional and analytical. Acquisitional methods describe the process of interacting with the expert to obtain information, while analytical methods describe how we use the information to derive rules. Each of these two methodological components has two subclasses (see Fig. 3).

As shown in Fig. 3, acquisitional methods consist of either observational or introspective approaches. In the observational approach, we watch the expert solving actual or simulated problems. If possible, we have him/her describe their

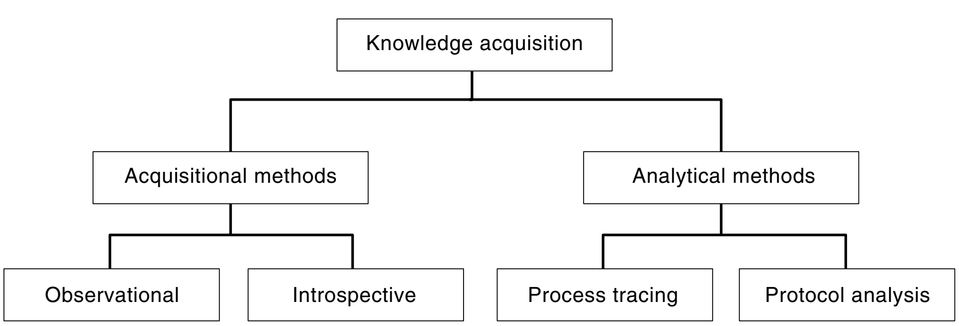

Figure 3. Basic classification of knowledge acquisition methods.

solution approach as they go through it. The introspective approach has the expert respond to examples provided by the knowledge engineer. The expert then describes in detail how problem solving occurs for the examples. Clearly, these two approaches are not mutually exclusive, and the knowledge engineer can frequently employ both to obtain the need information for expert system development.

Information acquired from the expert must be converted into rules. Process tracing takes the transcript of the session with the expert and looks for paths from data to decisions. Protocol analysis is a more detailed look at the transcript and also other relevant information about the problem-solving situation. To develop protocols, we look for inputs to decision making. We also look for relevant nonverbal data and background knowledge. In a sense, protocol analysis can begin with process tracking and then expand to acquire additional information. Once we have developed the protocol, we look for important elements of the protocol beyond informational elements that would change the problem solving procedures. For instance, does order matter? When did the alternatives develop and when did attributes get instantiated? Answers to questions such as these help to convert protocols into knowledge.

Converting protocols into rules is the final phase of knowledge acquisition. In some cases, the protocol analysis provides easily interpreted If-Then statements. In other cases, additional work is needed. One tool for accomplishing this is the repertory grid. The grid consists of two parts, constructs and elements. Constructs are the attributes or informational characteristics obtained from the protocol analysis. Elements are key examples that the knowledge engineer hopes to use to clarify the rules. The knowledge engineer, with the help of the expert, then looks for groupings of the examples based on the constructs. These groupings define the constructs or attributes that are used for problem solving in the examples. For instance, consider a medical diagnosis problem. We may have a variety of different problem-solving approaches for the patients. By examining the constructs or attributes, we may find that age of the patient is an important construct in determining which protocol is initiated. Repertory grids provide a convenient method for performing this analysis and, hence, converting protocol information into rules [for more details see (29)].

Because knowledge acquisition is the major bottleneck in constructing expert systems, a number of researchers have built tools for the process. These tools essentially help to bring knowledge directly from the expert into rules. However, most expert systems still require considerable work by knowledge engineers. A good description of this knowledge acquisition research is provided by (30).

Knowledge Representation

The third phase in expert system development is knowledge representation. The major objective in this phase is to take the acquired knowledge and translate it into machine-readable form. There are many different methods of knowledge representation in expert system development, and in this section, we discuss the two most popular ways to represent knowledge: rules and frames. For a discussion of other knowledge representation forms, see (11,16,17,18). The focus of the first part of this discussion on knowledge representation is knowledge that is stated in a deterministic state. In a later section, we provide a discussion of the modes used to represent uncertain knowledge within a knowledge base.

Rules. Currently, the most popular method of knowledge representation is in the form of rules (also known as *production rules* (22) or *rule-based systems*).

In Fig. 4, we illustrate the use of rules through a simple rule base of five rules created for the domain of credit approval. There are many questions a loan officer may ask in the process of deciding whether to approve or deny an application for credit. Some of the questions the officer may ask concern

- The current salary of the person
- The credit history of the person
- Their current employment

A simple (fictitious) rules base that might be applicable to this domain is given in Fig. 4.

One of the first things to notice about the representation of the knowledge is the simple structure of the rules themselves. The knowledge of the decision-making process is given in the form of simple IF-THEN constructs. Note that each of the rules contains one or clauses in the IF part of the rule; these clauses are known as the *antecedent,* and one (but po-

```
RULE NUMBER: 1
IF:
        The customer's income is less than 25,000.
THEN:
        The customer's line of credit has been approved: no.
------------------------------------------------
RULE NUMBER: 2
IF:
        The customer's income is at least 25,000.
AND     The customer's rating is excellent.
THEN:
        The customer's line of credit has been approved: yes.
------------------------------------------------
RULE NUMBER: 3
IF:
        The customer's income is at least 25,000.
AND     The customer's rating is good.
AND     The customer has been in their present job less than 2.5 years.
THEN:
        The customer's line of credit has been approved: no.
------------------------------------------------
RULE NUMBER: 4
IF:
        The customer's income is at least 25,000.
AND     The customer's rating is good.
AND     The customer has been in their present job at least 2.5 years
THEN:
        The customer's line of credit has been approved: yes.
------------------------------------------------
RULE NUMBER: 5
IF:
        The customer's income is at least 25,000.
AND     The customer's rating is poor.
THEN:
        The customer's line of credit has been approved: no.
```

Figure 4. An example rule base for the loan application problem.

tentially more than one) clause in the THEN part of the rule; these clauses collectively are called the *consequent*.

In each of the rules in Fig. 4, the antecedent of each rule contains n clauses (all joined by AND) that must *all* be true for the rule to become *triggered* (added to the conflict set). The process of instantiating the consequent of the rule is known as *firing* of the rule. Formally, a rule is fired if and only if the antecedent of the rule is true, and the consequent is instantiated.

As can be seen from the rules in Fig. 4, the loan officer's first criteria for deciding whether to approve or deny the loan application is current income. That is, if the person's current income is less than $25,000, then they cannot be approve for the loan. However, if their income is at least $25,000, other conditions (such as credit history and, possibly, years on a job) must be checked in order to make this decision.

The popularity of rules as a mode of knowledge representation has occurred for many reasons. One advantage to using rules is their modularity. Each rule in the rule base potentially stands apart from the other rules. Additions and deletions of rules can be made easily. Care must be taken when adding or deleting rules because the logic of the decision-making has now been potentially changed.

A second advantage to the use of rules is their uniform structure. From the discussion and the formal representation given above, all rules in a rule base have the same form. Each rule contains one or more antecedent clauses (usually joined by an AND) and one or more consequent clauses joined by an AND.

Lastly, rules provide a natural mode of knowledge representation. The time required to learn how to develop rule's bases (knowledge bases that contain rules) can be kept to a minimum. In addition, many experts solve problems based on combining pieces of evidence (known facts), and the combination of those facts lead to other newly inferred facts (i.e., the consequent). Lastly, there exist many expert system development packages, known as shells, which use rules as the primary method of knowledge representation. Expert system shells will be discussed in more detail later.

While rules have many advantages over the other form of knowledge representation, they also have some drawbacks. A knowledge base of rules can quickly become unwieldy and unmanageable if not properly implemented. Thorough documentation of the individual rules and the rules that they most likely interact with must be kept. In addition, rules can be hard to maintain for the same reasons previously stated. Rules can be inefficient in processing because the inference engine is performing a search over the rules to find the rules that could be fired given the current state of knowledge in the system. Lastly, rules cannot handle all types of knowledge. There are many different knowledge representation modes that have been proposed, and while rules are suitable for most applications, there does exist certain types of knowledge for which it is not well suited.

While rules are currently the most popular means of knowledge representation, the formation of good rules is still more of an art rather than a science. The development of structured programming techniques for rules is given in (31,32).

Frames. The use of object-oriented methods in software development has impacted the development of expert systems as well. Knowledge in an expert system can also be represented using the concept of objects to capture both the declarative and procedural knowledge in a particular domain. In expert systems, the terminology that is used to denote the use of objects is *frames* (33), and frames are fast becoming a popular and economical method of representing knowledge. Frames are the earliest application of object-oriented technology and therefore provide many of the benefits that have been attributed to object-oriented systems. In this section, we discuss the basic elements of frames as a knowledge representation mode; a further, more detailed explanation is presented in (16).

A frame is a self-contained unit of knowledge that contains all of the data (knowledge) and the procedures associated with the particular object in the domain. In Fig. 5, we show a hierarchy of objects using the classification of humans as the particular domain. Each of the frames in Fig. 5 represents an object in the domain. The top-level object is known as the *class*. As you proceed down the tree, each of the objects become a more specific example of the upper node. For instance, Jack is a particular example of a Male and Human; we call

Class name : Human

Subclasses : Men, Women

Properties :

Age	Unknown
Number of legs	2
Residence	Unknown
Life expectancy	70

Class name : Men

Subclasses :

Properties :

Age	Unknown
Number of legs	2
Residence	Unknown
Life expectancy	64
Mustache	Unknown

Class name : Jack

Class : Men

Properties :

Age	45
Number of legs	1
Residence	Harrisonburg
Life expectancy	85
Mustache	True

Figure 5. A typical frame hierarchy exhibiting the object-oriented approach to knowledge representation.

Jack an *instance* of the class Human, while Male is a *subclass* of Human.

There are three basic types of frames that must be written in a frame-based system: a class frame, a subclass frame, and an instance frame; all of these are shown in Fig. 5. A class frame consists of all of the relevant attributes that pertain to the application at the highest level. In Fig. 5, the relevant attributes for the class Human are age, number of legs, resident, and life expectancy. Both the subclass and instance frames inherit all of the attributes from the class frame, and in addition, more specific attributes can be added. The basic difference between the three types of frames is the level of detail of the attributes, their associated values, and the placeholders that link the frames.

In addition, frames may have procedures (methods) associated with each of them. These procedures allow the frames to act on the data in the frame to make change/updates when necessary. Many times, frames are combined with rules in knowledge representation in order to capture the complexity of the domain (16).

Other Modes of Knowledge Representation. Rules and frames are not the only modes of knowledge representation that are available to knowledge engineers. In this section, we will briefly introduce some of the other modes currently being used for knowledge representation.

Logic, specifically predicate logic, is one of the oldest forms of knowledge representation. Predicate logic is based on the idea that sentences (propositions) express relationships between objects as well as the qualities and attributes of such objects (17). Within predicate logic, the relationships are expressed by *predicates,* and the objects themselves are represented by *arguments* of the predicate. Predicates have a truth-value, depending on their particular argument; specifically, predicates can either be true or false.

Cases, or case-based reasoning, represent a different level of abstraction from rules. A case encapsulates the entire problem description and solution in an object called a case. Inference involves defining features from the case and then retrieving the “best” matches based on these features. The matching of features can be quite complex.

Cases are used to capture the previous experiences of expert in solving problems in a domain. When presented with a new situation, the system attempts to match previous cases with the given situation. The previous cases are adapted in order to provide a solution for the given situation. More about case-based reasoning and the use of cases for knowledge representation can be found in Refs. 34 and 35.

Uncertainty Management

Up to this point, we have considered knowledge representation strategies under conditions of certainty. Very few real problems have this characteristic. Hence, we need to investigate methods for representing problem solving knowledge under conditions of uncertainty. Despite considerable research activity, reasoning under uncertainty remains difficult because of the desire for both rigorous and easy to apply methods. Unfortunately, these two objectives turn out to be conflicting in the domain of uncertainty management. The most rigorous and justifiable methods are also the most difficult to implement. Conversely, the most commonly implemented techniques have little, if any, theoretical underpinnings. Hence, the knowledge-based system designer must carefully weigh the trade-offs in his or her particular situation and choose an approach to uncertainty management based on these trade-offs. In this section, we will discuss the major approaches to managing uncertainty in expert systems. As we discuss each of the approaches, we will highlight their major strengths and weaknesses with a view to providing the reader with the capability to make critical assessments.

Before proceeding with a description of the approaches to uncertainty management, we need a clearer picture of the nature of uncertainty as it affects knowledge-based systems. Suppose we have represented problem solving knowledge in the following rule: IF pulse is thready and foot skin temperature is low, THEN cardiac index is low.

This rule represents a model of problem solving reality typically taught to nurses and attending physicians in an intensive care unit. But like many problem-solving models, it provides only an approximate representation of a complex reality. As we examine this model from the standpoint of uncertainty management, we note several sources of uncertainty. First, the rule itself encapsulates an uncertain relationship. Not every person with these conditions has low cardiac index. The lack of precision or uncertainty in this rule is typical of problem solving rules or models. Most are approximations to reality with some error associated with their input to output mappings.

A second source of uncertainty concerns the evidence in the antecedents of the rule. We may not know for certain that the pulse is thready, because this evidence might come from a trainee who is inexperienced evaluating a pulse measurement. Further, even experienced nurses might differ among themselves about subjective measurements, such as this. Hence, even if we believe the rule contains little uncertainty, the evidence itself might be highly uncertain.

Finally, we note that the terms used in this rule have uncertain meaning. The rule is written in a fashion consistent with the training given to intensive care nurses who can acquire more understanding of terms like “low” and “thready” through extensive training. However, computer-based approaches, such as expert systems, require structured algorithmic methods to handle the uncertainty in natural language statements. As we shall see later in this section, some investigators differentiate between uncertainty and imprecision. They argue that natural language statements contain imprecision instead of uncertainty and should, therefore, be handled with different mechanisms.

This section will provide an overview to the major approaches to uncertainty management for expert systems. We will explore the basic mechanisms for reasoning under uncertainty advocated by each approach and then consider their comparative strengths and weaknesses. While the field has produced many more approaches than the ones considered here, nonetheless, these remain the best known and used methods in existence. Other approaches tend to build on these for very specialized applications, and hence, can best be understood in the context of the more basic and general methods described here.

Bayesian Inference. Bayesian inference provides the foundation for the most formal and mathematically rigorous of the uncertainty management schemes used in expert systems. At the center of Bayesian inference is the notion of subjective probability. Traditional definitions of probability use frequentist arguments: the probability of an event is the frequency of occurrence of that event. Bayesian or subjective probability extends this definition to include personal measures of belief in the occurrence of an event (see Refs. 36 and 37). The arguments for and against this perspective are lengthy and beyond the scope our concerns here (see Ref. 38 for a detailed discussion). Rather, we take as given the arguments for subjective probabilities and the considerable axiomatic machinery that accompanies probability theory in general. We focus instead on the reasoning process necessary for using this approach as the basis for uncertainty management in expert systems.

As the name implies, the major tool for reasoning with probabilities according to Bayesian inference is Bayes rule. This rule, which follows directly from axioms about conditional probability, shows how to update the probability of an event given evidence about the occurrence of another related event. The rule is easily illustrated through an example.

Suppose we want to build an expert system to perform a diagnostic task (e.g., diagnose the cause of problems in a desktop computer). Suppose further that we have n mutually exclusive and exhaustive hypotheses about the causes of the problem in the computer. We label these hypotheses $H_1, \ldots, H_n$. By mutually exclusive, we mean that no more than one hypothesis can be true. By exhaustive, we mean that at least one hypothesis must be true. Hence, exactly one among the set of hypotheses we will code into our expert system must be the true cause of any problem we will present.

This assumption appears quite daunting for expert systems developers, and it should. However, since we can define the hypotheses in any way we desire, we can always create a "none of the above" hypothesis that accounts for all other causes. This approach can many times effectively handle the exhaustic part of the assumption. The mutually exclusive part is more difficult to treat, and we will postpone our discussion of approaches until we have successfully presented a more thorough foundation for Bayesian inference.

For our system to reason effectively about the hypotheses, we will require evidence. Different domains have quite different types of evidence, but most expert systems work with evidence that comes from finite sets. Bayesian inference is not limited to this group, and the interested reader can consult (39) to see the details of extending the approach here to continuous or infinite domains.

Suppose that our evidence consists of the outcome of another related event (e.g., the presence of certain information on the screen of our troubled computer). If there are m possible outcomes for this event than we can label these $X_1, \ldots, X_m$. Given the evidence that outcome X_j occurred, our goal is to find the (a posteriori or more simply posterior) probability of each hypothesis or $\Pr\{H = H_i|X = X_j\}$ for $i = 1, \ldots, n$. To find these, we need a formal relationship between the evidence and the hypotheses. This relationship is given by conditional probabilities: $\Pr\{X = X_j|H = H_i\}$ for $i = 1, \ldots, n$ and $j = 1, \ldots, m$. Because we know the value of X and want these probabilities as a function of H, we call these conditional probabilities the likelihood functions. Finally, we also need to know the probability of each hypothesis before observing the evidence. We label these hypotheses priors and use the notation $\Pr\{H = H_i\}$ for $i = 1, \ldots, n$.

Once we have the prior probabilities, the likelihoods (or conditional probabilities for X given a value for H), and the occurrence of a specific X, we can use Bayes rule to provide us the probability for each hypothesis given the evidence. Bayes rules is

$$\Pr\{H = H_i|X = X_j\} = \frac{\Pr\{X = X_j|H = H_i\}\Pr\{H = H_i\}}{\sum_{i=1}^{n} \Pr\{X = X_j|H = H_i\}\Pr\{H = H_i\}}$$

Before going any further, we need to see how to use this rule in expert systems. For our computer diagnosis example, suppose our computer fails to dial using its modem. For simplicity, we consider three hypothetical sources for the problem: the modem (H_1), the modem or device controller software (H_2), and the communications software (H_3). A priori, we think each of these hypotheses is equally likely, so $\Pr\{H = H_i\} = 1/3$ for $i = 1, 2, 3$.

Now, suppose as evidence we successfully reinstall the controller software, and we still have the modem still fails to connect. So our evidence, X, the state of the machine after the reinstallation is continuing failure to operate (X_1) rather than successful operation (X_2). Note that while a successful operation at this point would allow us to conclude with high probability that the original defect was in the controller software, the continued failure does not allow us to exclude the controller software from further consideration. Let $\Pr\{X = X_1|H = H_i\} = 1$ for $i = 1$ or 3 and $\Pr\{X = X_1|H = H_2\} = 0.2$. The first of these probabilities says that we believe the modem will fail to connect after the software reinstallation if the problem is either a defective modem or communication software.

Using this information and Bayes rule, we can easily find that $\Pr\{H = H_i|X = X_1\} = 0.455$ and $\Pr\{H = H_2|X = X_1\} = 0.09$. Hence, the probability that the controller software is defective has dropped to less than 10%, while the probability for each of the other two hypotheses has increased to almost half. A rule based system that contained the following rule

> IF no connection after reinstallation of controller software THEN modem defective or communications software defective

would allow us to reach its conclusion with a probability of about 0.91.

Notice first that our use of Bayes rule in this example has provided a mechanism for handling the uncertainty inherent in the rule. We have not seen how to handle uncertainty in the evidence. Further, in this simple example, we only reasoned through one level. That is, we collected our evidence, fired one rule, and reached our diagnosis. In most problems, we want to handle more complex forms of reasoning that involve multiple types of evidence.

It turns out that we can treat both of these issues in exactly the same way. Suppose that an acquaintance performed the controller reinstallation and then reported to us that the modem still would not dial. Because this person is not as skilled as we are, we are reluctant to conclude with certainty that $X = X_1$ as we did before. To handle this situation, we add another layer to our reasoning and call this new evidence Y. We let $Y = Y_1$ if the friend reports that the reinstallation and test failed to correct the problem and $Y = Y_2$ otherwise. Our X variable has had subtle change of meaning. Rather than the actual result of our test, it now reports the result we would get if we did the reinstallation and test rather than our friend. A priori, we might believe the probability that our reinstallation test would show failure is slightly less than the probability that the controller software is the problem. So we assign $\Pr\{X = X_1\} = 0.3$. The conditional probability for our friend's result, $Y = Y_1$, given we know our result measures our confidence in our friend's test and, hence, the evidence. Suppose we evaluate it as $\Pr\{Y = Y_1|X = X_1\} = 0.9$, and also $\Pr\{Y = Y_1|X = X_2\} = 0.2$. Then, applying Bayes rule, we get $\Pr\{X = X_1|Y = Y_1\} = 0.63$ or slightly more than double what we believed a priori.

To see how we use Bayesian inference to chain rules and evidence together, we can now calculate the new probability for the hypotheses concerning our defective modem. To do this, we need to find $\Pr\{H = H_i|Y = Y_1\}$. We will again use Bayes rule, but to do this, we must obtain the likelihoods or $\Pr\{Y = Y_1|H = H_i\}$. We obtain these using the law of total probability, expressed in this case as

$$\begin{aligned}\Pr\{Y = Y_1|H = H_i\} &= \Pr\{Y = Y_1|X = X_1\}\Pr\{X = X_2|H = H_i\}\\ &+ \Pr\{Y = Y_1|X = X_2\}\Pr\{X = X_2|H = H_i\}\end{aligned}$$

Inserting the values given for the quantities on the right side of this expression, we obtain $\Pr\{Y = Y_1|H = H_i\} = 0.9$, 0.34, and 0.9 for $i = 1$, 2, and 3, respectively. We can now put these values into Bayes rule in combination with our prior probabilities for the Hi, $i = 1, 2, 3$. The resulting values for the hypotheses are $\Pr\{H = H_i|Y = Y_1\} = 0.42$, 0.16, and 0.42 for $i = 1$, 2, and 3, respectively. Notice that the uncertainty in our evidence has now increased the probability of H_2, given the evidence, by more than one and one half times its value when the evidence was certain. In other words, we are much less confident that we can discard the controller as a cause for the problem when our test contains some uncertainty.

Recently, several authors [for example, (40)] have proposed methods for applying Bayes rule to knowledge-based systems that use local calculations at each rule. The above approach required us to have knowledge of probabilities stored in other rules, as is evident in the total probability calculation. These methods employ rules and evidence structured as directed, acyclic graphs or DAGs. In order to apply local computations in this scheme, we need one additional assumption—conditional independence. The version we illustrate here, called Markov chain independence, has the form

$$\Pr\{X, Y, Z\} = \Pr\{Z|Y\}\mathrm{P}\{Y|X\}\mathrm{P}\{X\}$$

The DAG for this rule is shown in Fig. 6.

To reason with this DAG, we use series of updating rules. The details for these rules are given in (40). This approach has significant computational advantages over more traditional applications of Bayesian inference while maintaining the formal theoretical basis for the procedure.

The above examples and discussion provide insight into the major advantages and disadvantages of Bayesian inference for uncertainty management. Among its primary advantages is the fact that Bayesian inference provides a formal, rigorous quantification of uncertainty. This also means that both users and experts have a precise interpretation of probabilities as subjective measures of uncertainty. So when an expert says that an event has a 0.9 probability of occurrence, this means that he or she would place a bet on that outcome according to that probability. No other approach to uncertainty management has this clear interpretation.

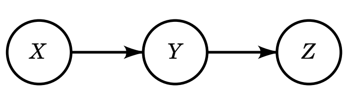

Figure 6. Example of a simple directed, acyclic graph.

However, we pay a price for the formality and precision of Bayesian inference. While the use of local computations in directed, acyclic graphs has somewhat reduced the computational burdens, this approach still has greater computational complexity than several of its competitors. More important than the computational problems are the assessment burdens. For a rule that considers three hypotheses and four event states, we need to obtain 12 conditional probabilities and three prior probabilities. Essentially, this means about an order of magnitude more probability assessments than we have rules. Further, if we need to build any of these probabilities from data collection, we need an order of magnitude more data than probabilities. For expert systems with hundreds or thousands of rules, the probability assessments can become extremely burdensome.

Another equally important disadvantage for expert systems development is that Bayesian inference does not allow for incremental development. Because we need to specify all the hypotheses and evidence in order to apply the updating rules, we cannot incrementally add rules as we build the system. Instead, we must specify all components before we construct the uncertainty management system. This works against the development philosophy for expert systems, which seeks to incrementally add rules and change rules in consultation with the domain expert.

A final disadvantage cited by some to Bayesian inference is the treatment of mutually exclusive and exhaustive events. In order to perform our calculations, we need to define this set of events and assign appropriate probabilities. Critics have argued that in some cases, this is not appropriate. This criticism is debatable since from a Bayesian perspective, this criticism can be handled by redefining the outcome space to correspond to the current state of knowledge. Nonetheless, this criticism has lead to the development of a competing approach to uncertainty management that we will consider in the next subsection.

Dempster–Shafer Theory of Belief Functions. As we noted in the previous section, Bayesian inference requires a mutually exclusive and exhaustive set of alternatives for any outcome. This means that the probabilities for these outcomes must sum to one. Some have argued that this requires people to express greater certainty than they actually have in an outcome. For instance, in our previous example, we expressed prior probabilities of 1/3 for each hypothesis about the cause of our modem problem. Suppose we really do not know the cause and feel that 1/3 is too high for any of these hypotheses. However, any lower value would violate the sum to unity requirement for probabilities for mutually exclusive and exhaustive events.

Shafer (41) proposed an approach to uncertainty management that allows for expressions of this type of uncertainty. According to this theory, we assign a degree of belief denoted Bel to a possible outcome or hypothesis. Bel(A) measures the strength of belief or evidence in favor of hypothesis A and takes on values from 0 (no evidence) to 1 (certainty).

In addition to Bel, this theory also defines a concept called Plausibility or Pl. Pl(A) measures the belief or evidence for hypothesis A when we remove Bel($\neg A$) where $\neg A$ is negation of the hypothesis A. So, Pl(A) = 1 − Bel($\neg A$). The range [Bel(A), Pl(A)] forms an interval for the probability mass of A, m(A). The size of this interval gives us a measure of our uncertainty regarding the probability of A.

To assign probability mass in our particular problem domain, we start by considering the set of possible hypotheses, which in the theory of belief function is called the frame of discernment or θ. For our computer modem problem, there are three hypotheses, so $\theta = \{H_1, H_2, H_3\}$. Rather than assign probability to this set, the theory of belief functions allows assignment of probability to each member of the power set (or set of all subsets) of the θ. In this case, this consists of eight or 2^3 subsets including the empty set and θ. The more mass we assign to θ, then the more uncertain we are about the probabilities. So if we set $m(\theta) = 1$, we have no information about which of the three hypotheses might be true (hence, Bel(H_i) = 0 and Pl(H_i) = 1 for i = 1, 2, 3). As we become more confident in our probability assignments for the hypotheses through the accumulation of evidence, then the value for $m(\theta)$ decreases toward 0.

To accumulate probabilities in hypotheses, the theory needs a mechanism for updating probability mass values. This mechanism is provided by Dempster's rule. We can describe this rule by referring again to our example modem problem. Suppose we want conduct two tests; the first (test 1) is after reinstalling the controller software, and the second (test 2) is after reinstalling the communications software. The modem still does not work after each installation. Let m_1 and m_2 be our probability mass assignments after each separate test. Dempster's rule gives use a way to compute the effect of the combination of these two tests on the probability masses. Let his combined probability mass be m_c. So for hypothesis H_i, Dempster's rule is

$$m_3(H_i) = \frac{\sum_{A \cap B = H_i} m_1(A) m_2(B)}{1 - \sum_{A \cap B = \varnothing} m_1(A) m_2(B)}$$

As with Bayesian inference, the Dempster–Shafer theory of belief functions has its advantages and disadvantages. The primary advantage is the capability to describe in greater detail uncertainty about hypotheses. However, the price one pays for this added feature is a considerable increase in the computational and assessment burden. Where the assessment for Bayesian inference was considered difficult, for the theory of belief functions, the assessment burden has grown exponentially ($2k$) in the size of each set of alternatives (k). Obviously, this explosive growth also adds to the computational burden. Further, unlike Bayes rule, Dempster's rule of combinations is a heuristic which has no theoretical justification other than equivalence to Bayes Rule under equivalent conditions. In conclusion, the Dempster–Shafer theory of belief functions has addressed one issue with Bayesian inference but at the cost of making all other concerns much worse.

Certainty Factors. Shortliffe and Buchanan (9) in their development of one of the first expert systems proposed certainty factors, MYCIN. They wanted to develop a computationally tractable approach to handling the uncertainty involved in recommending treatments for patients with bacterial infections. They also wanted a method that provided for easy assessments of uncertainty and modifications when new rules where added to the knowledge base. Finally, as with the Dempster–Shafer approach, Shortliffe and Buchanan were also interested in describing situations that contained an incomplete set of hypotheses.

Shortliffe and Buchanan define the certainty factor for hypothesis H given evidence E, CF(H, E), as the difference between measure of belief in H given E, MB(H, E), and the measure of disbelief in H given E, MD(H, E). They defined these two quantities in terms of the conditional probabilities as shown:

$$\mathrm{MB}(H,E) = \begin{cases} 1 & \text{if } \Pr(H) = 1 \\ \dfrac{\mathrm{Max}\{\Pr(H|E), \Pr(H)\} - \Pr(H)}{1 - \Pr(H)} & \text{otherwise} \end{cases}$$

$$\mathrm{MD}(H,E) = \begin{cases} 1 & \text{if } \Pr(H) = 0 \\ \dfrac{\Pr(H) - \mathrm{Min}\{\Pr(H|E), \Pr(H)\}}{\Pr(H)} & \text{otherwise} \end{cases}$$

With these definitions, we note that the range for both MB and MD is 0 to 1. MB is 0 when the evidence fails to support the hypothesis, and MD is 0 when the evidence supports the hypothesis. Since CF(H, E) = MB(H, E) − MD(H, E), then the range for CF is −1 to 1.

We now need to provide a mechanism for combining evidence in rules. Consider first the situation where two rules provide evidence for a single hypothesis, H. Denote the evidence in each rule E_1 and E_2. Then, we find the measures of belief and disbelief from both pieces of evidence as

$$\mathrm{MB}(H, E_1 \wedge E_w) = \begin{cases} 0 & \mathrm{MD}(H, E_1 \wedge E_w) = 1 \\ \mathrm{MB}(H, E_1) + [1 - \mathrm{MB}(H, E_1)\mathrm{MB}(H, E_2)] & \\ & \text{otherwise} \end{cases}$$

$$\mathrm{MD}(H, E_1 \wedge E_w) = \begin{cases} 0 & \mathrm{MB}(H, E_1 \wedge E_w) = 1 \\ \mathrm{MD}(H, E_1) + [1 - \mathrm{MD}(H, E_1)\mathrm{MD}(H, E_2)] & \\ & \text{otherwise} \end{cases}$$

To illustrate this idea, consider again our computer problem diagnostic system. Our specific problem is to diagnose the cause of our inability to dial over a commercial telephone line using our computer and its installed modem. One of our rules assigns a measure of belief of 0.4 to the hypothesis that the modem is at fault, H_1, given the failure of dial-up test after reinstalling the controller software. Another rule assigns a measure of belief of 0.5 to this same hypothesis given the failure of a dial-up test after we reinstall the communications software. Hence, the measure of belief given both pieces of evidence is MB(H_1, E_1, E_2) = 0.4 + 0.6 · 0.5 = 0.7. Notice that the order in which the evidence is presented does not matter.

Now, consider the case in which rules are chained together so that the uncertain outcome of one rule feeds into another rule. This is similar to the uncertain evidence example we considered in our discussion of Bayesian inference. Suppose that we receive evidence E_1 about the outcome of our modem test from a less than completely reliable source. Let S be the source of our evidence E_1, and we assign a certainty factor to E_1 given S: $\mathrm{CF}(E_1, S) = 0.8$. Then, we can find our measure of belief given the evidence from this source as $\mathrm{MB}(H_1, S) = \mathrm{MB}(H_1, E_1) \cdot \mathrm{Max}\{0, \mathrm{CF}(E_1, S)\} = 0.4 \cdot 0.8 = 0.32$.

Finally, we need a method to conjunctions and disjunctions of hypotheses. This allows us to have rules with predicates that have conjunctions and disjunctions. The combination rules are

$$\mathrm{MB}(H_1 \wedge H_2, E) = \mathrm{Min}\{\mathrm{MB}(H_1, E), \mathrm{MB}(H_2, E)\} \text{ and}$$
$$\mathrm{MB}(H_1 \vee H_2, E) = \mathrm{Max}\{\mathrm{MB}(H_1, E), \mathrm{MB}(H_2, E)\}$$

Certainty factors provide a convenient way to model uncertainty in an expert system. They correct for many of problems we observed in Bayesian inference. In particular, they provide quickly computable solutions through a set of simple combination rules as given above. Because the certainty factors are associated with rules and separated from the methods of combining evidence, this allows easy extensibility of the rule base. Also, the assessment burden is greatly reduced to one assessment per proposition in a rule. This is clearly much more manageable than the large numbers of assessments for Bayesian inference or the exponential growth in assessments for the Dempster–Shafer approach.

The major drawback of certainty factors is their lack of theoretical or formal foundation. Despite their definition in terms of conditional probabilities, their use can lead to strange and uninterpretable results. Consider for example a problem with three mutually exclusive hypotheses. The first two are equally likely with probabilities of 0.49, while the remaining hypothesis has a probability of 0.02. We now obtain evidence that completely excludes the third hypotheses but gives us no information about how to discriminate among the other two. Hence, $\Pr(H_i|E) = 0.5$. Using our formula for the measure of belief, we obtain

$$\mathrm{MB}(H_i, E) = [0.5 - 0.49]/[0.51 \quad 0.5] = 0.04 \text{ for } i = 1, 2$$

which is an unacceptably low value given the value for $\Pr(H_i|E)$. Further, even though we know that these two hypotheses are the only possibilities for this problem, we get

$$\mathrm{MB}(H_1 \vee H_2, E) = \mathrm{Max}\{0.04, 0.04\} = 0.04 \neq 1$$

Unfortunately, there are many more of these types of problems that one can encounter while using certainty factors. Additionally, while the probabilities used in Bayesian inference have an understandable interpretation, the definition of a certainty factor does not lend itself to an easily interpretable value.

Hence, while certainty factors have addressed many of the problems with Bayesian inference, they have lost both the rigor and interpretability in the process. The lack of rigor may not matter for systems built for applications where safety or critical performance is not an issue.

Fuzzy Sets. The last approach to uncertainty management does not actually address uncertainty at all. Fuzzy sets have been proposed for use in expert systems to address imprecision rather than uncertainty. Many expert systems use rules based on natural language expressions such as the patient's pulse is thready. We could, of course, model evidence of a thready pulse as uncertain and use one of the three previously described approaches. However, advocates for fuzzy sets argue that terms such as thready are inherently imprecise not uncertain, and should be modeled with different methods. Fuzzy sets represent a method for handling imprecision.

Fuzzy sets were first proposed by Lofti Zadeh (42) as an approach to modifying the notion of strict set membership. In traditional mathematics, a set either contains or does not contain a specified element. For example, the set of integers clearly contains the number 4 but does not contain the number 4.5. In proposing fuzzy sets, Zadeh argued that some sets are not as crisp as the sets of integers. For example, while many would agree that Abraham Lincoln would qualify for membership in the set of tall former heads of state, they might argue against Napoleon Bonaparte's membership in this group. The argument would center on the word "tall." By itself, this word does not admit a precise meaning. Hence, Zadeh argued that we should allow for membership functions for elements of sets that take on a continuum of values between 0 and 1. A value of 0 indicates the element is not a member of the set, while a value of 1 indicates membership. Intermediate values show a degree of membership in the fuzzy set.

Combination rules provide us with a way to join sets. While the field of fuzzy sets has explored a wide variety of combination rules, those provided initially by Zadeh remain the most popular. Let $m_A(x)$ denote the degree of membership of element x in a fuzzy set A. Then,

$$m_{A \cup B}(x) = \mathrm{Max}\{m_A(x), m_B(x)\}$$
$$m_{A \cup B}(x) = \mathrm{Max}\{m_A(x), m_B(x)\}$$

We can use fuzzy sets in expert systems as an approach to quantify the imprecision in the rule premises. One approach is to obtain membership values at the time of rule construction. We then reason with these values as rules are instantiated and fired. Another approach is to obtain membership values from the user at run time. For example, we might ask the nurse using our expert system to enter the membership value for the patient in the set thready pulse. Finally, since set theory forms the foundation for logic, we can also employ fuzzy logic. In fuzzy logic, the truth-values for a proposition take on a continuum of values between 0 and 1. Fuzzy logic provides a vehicle for reasoning with fuzzy extensions to the propositional calculus and the first order predicate calculus.

Because fuzzy sets represents an approach to imprecision rather than uncertainty, we cannot directly compare it with the other methods. However, we can make some general remarks that contrast the approaches. As with certainty factors, fuzzy sets provide a numerical value that lacks easy interpretation. This lack of interpretability is evident when we consider membership in the union of two complementary sets. Suppose we have a patient whom we assign membership of 0.6 in the fuzzy set, thready pulse. Their membership in the fuzzy set, not thready pulse, is 0.4. But according to our combination rule, the patient's membership in the union of these two sets is 0.6 not 1.0.

Fuzzy sets do possess many of the advantages of certainty factors, such as ease of assessment and computational tractability. However, unlike certainty factors, fuzzy sets were not designed to work with expert systems. So there is no one accepted approach for applying them in rule based systems.

Implementation

The implementation of an expert system is the process of taking the knowledge that has been acquired and represented—in rules, frames, or another mode—and putting it into machine-readable format. That is, actually taking the knowledge and putting into some computer code. This can be accomplished three different ways:

- Using a conventional programming language
- Using a programming language design for Artificial Intelligence programs (PROLOG or LISP) or
- Using an expert system programming environment known as a shell

In this section, we will briefly discuss the use of conventional and AI-specific programming languages and then focus attention on the use of expert system shells. There are other organizational implementation issues that need to be addressed in order for the expert system application to be successful. These issues are not specifically addressed here, but the reader is encouraged to further explore these issues by examining (11,28).

Expert systems can be developed using conventional programming languages such as BASIC, Pascal, C, and/or C++. In recent years, developers have turned to using JAVA to develop expert systems solutions as well. There are, in addition, programming languages that are specifically designed for AI programming. The programming language PROLOG (PROgramming in LOGic) is an implementation of predicate logic for computing and is, therefore, a natural environment for using predicate logic to represent knowledge in a domain; many successful applications have been developed in PROLOG (19). LISP (LISt Processor) is a programming language that was developed by John McCarthy in 1956 to write AI-based programs, including expert systems. More information of the use and syntax of PROLOG and LISP can be found in many AI textbooks (5,19,20).

The problem with using a conventional AI-specific programming language is that you not only have to build the knowledge base, but you need to create all the structural components discussed previously in the programming language you are using. This added effort and additional complexity has caused many expert system developers to look to other methods for implementing expert systems.

The popularity of expert system development can be highly attributed to the creation of programming environments that allow nonprogrammers to implement expert system applications; these programming environments are commonly referred to as *shells.*

As stated earlier, expert system shells were first introduced after the development of the MYCIN expert system. The developers of MYCIN realized what a time-consuming task developing an expert system was, so they emptied out the knowledge based from MYCIN and were left with EMYCIN (Empty MYCIN). This allowed future expert system developers to concentrate their efforts on the development of the knowledge base and just plug in their knowledge base of rules into the shell.

There have been a number of advances in expert system shells since EMYCIN, and there now exist numerous software vendors that sell expert system shells. Many of the shells, like EXSYS (from Multilogic, Inc.), are primarily rule-based shells that allow for easy development of rule-based expert systems. Other vendors, such as Level 5, have developed object-oriented expert system shells—i.e., Level 5 Object—to promote more sophisticated means of performing knowledge representation. Level 5 Object allows a developer to use both frames and rules, as well as other features, to develop expert system applications. The number of vendors and software products is too numerous to detail here, but Durkin (16 provides an appendix that lists many of the commercial shells available.

In recent years, expert systems have been able to be applied to more real-time applications, such as command and control or manufacturing operations, through the use of real-time expert system shells. The two most popular general-purpose shells on the market are G2, developed by Gensym Corporation and RT-Works, developed by Talarian Corporation are two of the most popular real-time shells available on the market today.

Finally, the future of expert systems may find a place on the Internet and through Web pages. Many vendors, including Multilogic, Inc., are developing expert system shells that function as part of a Web page. In the future, more World Wide Web applications may be based on expert systems to collect information and provide recommendations to consumers.

Testing, Verification and Validation, and Evaluation

An important component of any software development effort is the testing and evaluation of the software system (solution) to ensure correctness of the outputs and user satisfaction with the product in solving the given problem. Since expert systems are software solutions to problems, then the importance of testing and evaluation cannot be minimized.

During the development of MYCIN, the developers were looking for a method to test and evaluate the output (advice) given by MYCIN in the domain. The developers performed three studies to test MYCIN by comparing the output of the MYCIN systems to the output of the doctors around whose knowledge the system was built (9). The authors state that new studies were undertaken because they felt that the results of the MYCIN T and E were being biased by the fact that the evaluators knew that the system outputs were being generated by a computer program. To alleviate this problem, the authors undertook a blinded study—one in which the evaluators did not know whether the results came from a colleague or MYCIN. However, in this report (43), the authors discuss the results of the blind evaluation but give few details on how the evaluators or the test cases were selected. The study did show that MYCIN "worked as well as the experts in the domain." This testing effort attempted to mimic the *Turing Test.*

As more expert systems were developed, different evaluation techniques were suggested. These techniques tended to fall into two classes. First, there are those authors that try to apply traditional software engineering techniques to the testing of expert systems. These authors (44–49), and many of the papers in (50), claim that traditional verification and validation techniques work with expert system testing and should be used more extensively.

The second group of authors view expert systems as different from conventional software systems and, therefore, conclude that new techniques for testing and evaluation must be developed. One of the most vocal of this group is Green and Keyes (51). These authors, discussing the verification and validation (V&V) of expert systems, state succinctly that "lack of understanding has created a vicious circle; V&V of expert systems is not done because nobody requires it. Nobody requires V&V of expert systems because nobody knows how [to do V&V of expert systems]. Nobody knows how to do V&V of expert systems because nobody has done it.

In four separate papers, authors (52–55) review the current state-of-the-art in performing verification and validation on expert systems and examine steps necessary to perform V&V on expert systems. While O'Leary (54) states that effective methods of validating a KB are critical, he finds that the current methods allow the developer to only look at the individual system components and not how they work together. In another paper, he outlines four major steps in performing validation of expert systems. These steps include ascertaining what the expert system knows, does not know, or knows correctly; ascertaining the level of expertise if the system; determining if the expert system is based on a theory of decision making in the particular domain; and determining the reliability of the expert system. O'Keefe et al. (52) view validation as a part of evaluation, which is a broader area that seeks to assess the expert system overall value. After outlining some basic concepts, O'Keefe et al. review some standard methods for performing qualitative and quantitative validation of expert systems. While they admit that their discussion has been descriptive in nature, they point out that prescriptive methodologies for performing validation of expert systems are needed.

All the literature to date points to the fundamental problem in the area of expert systems testing and evaluation; however, no one has yet attempted to solve the problem. Concerning expert systems, there is a lack of a formal framework for discussion of their verification, validation, testing, and—the more general problem of—evaluation; as well as a major lack of standardization of terminology (56). This terminology disagreement leads to confusion on various methods.

Two important aspects in the testing of expert system software have been mentioned: completeness and consistency. Completeless is defined as the coverage of the domain by the particular expert system. In other words, does the expert system solve most of the problems (give correct solutions for many of the inputs) in the domain of interest? Within completeness, the items that are checked for include dead-end rules, missing rules, and unreachable rules. Consistency, on the other hand, refers to a mathematical concept as applied to the antecedents and consequents of the rules in a knowledge base. Consistency checks for redundant rules, conflicting rules, subsumed rules, circular rules, and unnecessary antecedent conditions. More on completeness and consistency is discussed in (17).

Beyond these aspects, some authors (2,3,57) have attempted to formulate methods for reliability evaluation of rule-based expert systems. Reliability is one small piece of the testing and evaluation process within software systems. By attempting to solve this small piece of the larger problem, the authors are attempting a bottom-up approach to expert system testing and evaluation. In addition to the development of reliability estimation techniques, the use of the test results and reliability information is used to enhance the design of the rules in the expert system (2,3,57). These efforts are ongoing, and further experience with numerous examples of expert systems needs to be performed.

Still today, many of the methods used to test and evaluate expert systems are either ad hoc or based on traditional software engineering methods. These methods may one day prove to be useful for the testing and evaluation of expert systems. However, at this point, new methods for finding the reliability of a KB in an expert system must be explored—especially in the context of a rapid prototyping development methodology.

The development of software metrics is a viable, and often the only, way to measure the progress of a software system currently in development. Design metrics are the most promising type of knowledge base metrics because they aid the expert system developer before coding has begun. Software complexity can also help software designers in making simple modifications that will aid in the understanding and testing of the system, and eventually, improve the reliability. Metrics for expert systems are at their infancy, and there is hope that metrics can be developed to aid an expert system developer during the process of building a KB. Design metrics and the early estimates of the reliability will aid the KB community more in producing more reliable and efficient systems.

Maintenance

Maintenance is often a major issue for any software system. This is even more true for expert system technology. Much of the knowledge in an expert system is (or potentially can be) changing constantly, and these knowledge units need to be updated. This problem of maintaining an evolving knowledge base has been referred to sustaining the knowledge base rather than maintaining. Sustenance of an expert system requires a steady upkeep of the rules (or whatever knowledge representation modes is used) (11). However, caution must be enforced when changing the knowledge in a knowledge base. The effects of a change in one part of the knowledge base can have devastating side effects in other parts. Care must be taken in order to ensure that the total knowledge of the knowledge base has been upgraded and not degraded by changes. Methods for performing knowledge base maintenance need to be developed in order to ensure knowledge integrity.

The development of an expert system is a complex and time-consuming process. Much research must still be performed on specific areas of the development process. However, this does not preclude the on-going development of expert systems today. In the next section, we highlight the development efforts of current systems.

CURRENT APPLICATIONS

As has been pointed out in many places in this article, expert/knowledge-based systems have a very broad applicability to decision problems in business, industry, government, the sciences, and even everyday life. In this section, we discuss some of the current applications of expert system technology and how governments and businesses throughout the world are using them. This section will include a discussion of how some major corporations are using expert system technology to improve operations and decision-making and, in turn, profitability. We then discuss the use and application of expert system technology in the international arena. Finally, we will look at some of the latest, most innovative applications of expert systems as presented at recent conferences on *Innovative Applications of Artificial Intelligence*—sponsored by the American Association for Artificial Intelligence (AAAI)—and the Third World Congress on Expert Systems (Seoul, South Korea).

Corporate Usage of Expert Systems

Not all companies have a major success story to tell—like that of XCON for Digital Equipment Corporation—when it comes to the application of expert system technology. However, many (small and large) corporations are finding key applications that save time and money by helping to make better, more consistent, and faster decisions. One of the major companies to embrace expert system technology is DuPont. Led by the efforts of DuPont's AI division director, Ed Mahler, DuPont began using expert system technology on many small applications. In particular, it was Mahler who instigated the deployment of well over 200 expert systems. Each of these systems is quite small—averaging about 80 production rules. However, Mahler estimates that aggregate savings to DuPont was at tens of millions annually (58).

The corporate strategy toward expert system development is not always small, but any small systems exist. For example, Boeing Corporation, the aerospace giant, uses a 25,000 rule, written in PROLOG, expert system to advise employees in the proper assembly of complex electrical connectors and cables for airplane manufacturing, maintenance, and repair (19). In addition, automobile manufactures such as Chrysler use expert systems for design of automobile cooling systems, and General Motors uses expert systems for diagnosing problems in manufacturing equipment (19).

Expert system technology is not limited to the manufacturing sector. American Express Corporation uses expert system technology to examine transactions and attempt to detect patterns of fraudulent card use. American Express' Authorizer Assistant (AA) is a rule-based system that provides the first line of service for credit authorization at the point of sale (19).

It is truly difficult to track the deployment of expert system technology in many companies due to the fact that many, most likely, include some form of proprietary information. Durkin's catalog of applications cities 2,500 working expert systems, but he estimates that the total number of expert system applications is easily over 25,000 systems (12).

International Usage

The use of expert system technology is not limited to only the United States. Organizations, academic institutions, corporations, and governments around the world have applied expert system technology to solve everyday decision problems.

This is most evident from the papers and tutorials that have been presented at the three, soon to be four World Congresses on Expert Systems. The World Congress on Expert Systems was established "to bridge the gap between the academician and the practitioner and concrete on expert system work being performed throughout the world" (59). Liebowitz goes on to point out that the congress tries to connect expert system theory and practice and promote the sharing of worldwide ideas. The congress usually has three major components: (1) expert system technology, (2) expert system applications, and (3) management of expert system programs and projects. The congress has attracted representatives from 45 countries and the past three congresses—Orlando, FL, 1991, Lisbon, Portugal, 1994, and Seoul, South Korea, 1996—have included close to 800 papers from about 50 countries. The Fourth World Congress on Expert Systems is due to take place in Mexico City in March 1998.

Medsker and Liebowitz (27) list a number of applications done in Europe, the Far East, Mexico, and Canada. European applications include expert systems for railway control (in France and Austria), a system for treatment of cases concerning the import and export of sugar products (Belgium), a system for controlling experimental sites in high-energy physics (Switzerland), and a system, called RAP, for naval resource allocation (England). In Japan, the focus is on manufacturing applications; however, an expert system for cockpit crew scheduling has been built for Japan Airlines. In North America, an expert system (RHUTA) to assign human resources to planned substations and transmission lines of a power network was built in Mexico, a system that provides personal information on how to reduce a person's risk of developing cancer was developed in Canada, and an expert system (VARMINT) for aiding maintenance and repair of machines on icebreakers was also developed in Canada. These are just a sampling of expert system applications in use around the world. In addition to these systems, many applications in the telecommunications industry worldwide are highlighted in (60).

Innovative Applications

Each year, since 1989, the American Association for Artificial Intelligence (AAAI) has sponsored an annual conference that highlights the most innovative applications of AI technology. The Innovative Applications of Artificial Intelligence Conferences were formed "to highlight the successful transition of AI technology from theory to practice, recognize AI applications and AI applications developers as integral contributions and contributors to the AI field at a national conference, and provide a forum for the exchange of experiences and lessons learned in the heartland of the AI community" (61). An *innovative* application "is one in which AI technology had enabled solutions for problems not previously thought to be amenable to computational solutions" (61).

Over the past nine conferences, including the July 1997 conference in Providence, RI, the most significant item has been the extreme diversity of application areas—ranging from space and computing to business operations and manufacturing. These application areas mirror the applications areas cited in earliest section of this report.

Some of the most interesting applications presented at recent conferences in the expert system area include a bounced mail expert system (BMES) for the White House to diagnosis failures in electronic mail delivery (62) and Fannie Mae's Automated Mortgage Underwriting Expert System (63).

THE FUTURE FOR EXPERT SYSTEMS

The future for expert systems development is bright; however, there remain many obstacles that must be overcome in order for expert systems to truly flourish into a common problem-solving methodology. The current generation of expert systems is plagued by three major limitations: information brittleness, isolation, and static knowledge. In this section, we discuss the on-going efforts to extend the usefulness of expert systems and overcome the limitations.

In order to overcome the limitations inherent in the technology, methods of learning and the integration with other technologies must be incorporated in intelligent systems that solve critical problems in changing domains. In this section, we will discuss the use of expert systems embedded within other technologies, the use of hybrid intelligent systems as problem solvers, and the current state-of-the-art in learning mechanisms that can be incorporated in expert systems to overcome these inherent limitations.

Finally, we will discuss the applications and use of expert systems in the area of knowledge management and business process reengineering and the role of expert systems in distributed artificial intelligence and intelligent agent systems.

Embedded Systems

Artificial intelligence systems, including expert systems, can be broadly categorized into two general classes based on their architecture: stand-alone and embedded. Typically, an expert system has been developed in a stand-alone architecture and exists either independently or as the main component of a system that relies on another system for data collection (64).

An embedded expert system would be one that is designed and built to be an integral part of some larger system environment. The overall system environment provides a wide range of functions that support the system's mission and define its architecture. The embedded expert system can provide these functions directly, or support them indirectly as services. In either case, the use of an expert system should be invisible to the surrounding system and the user.

The future of expert systems will be as part of larger systems in an embedded architecture. Both software systems and consumer products will have expert systems' functionality embedded within the product, and that functionality will be invisible to the user (consumer). Current uses of expert systems as embedded systems are highlight in an *IEEE Expert* Special Issue on Embedded AI Technology in June 1994 (64).

Hybrid Systems

One of the major reasons for the rise of expert systems has been the failure of other traditional techniques to address problems of automating problem-solving knowledge. Approaches from operations research have attempted to optimize where, in many cases, optimization is not possible. On the other hand, recent interest in neural networks has shown where expert systems have failed to address important aspects of problem-solving knowledge acquired through inductive learning.

Instead of relying entirely on a single technology, many complex domains require multiple technological solutions. When combined into a single system, these hybrids can sometimes outperform the solutions provided by their individual technological components. For example, neural networks exploited small computational building blocks to achieve intelligent behavior. However, this raw computational approach tends to overlook problem-specific characteristics that could aid problem solving. The addition of rules can sometimes significantly enhance the performance of these systems. For example, rules that order the presentation of training instances in back propagation neural networks can significantly decrease training time. Also, rules that preserve diversity in genetic algorithms can enhance their performance. Hence, combinations of techniques from neural networks, operations research, statistics, and expert systems can provide powerful problem-solving methodologies. A number of examples of successful hybrids are described in (65–68).

Learning

Developing machines that learn, in the sense of biological systems, remains one of the fundamental challenges in artificial intelligence. Nonetheless, a great deal has been accomplished over the last several decades of research in this area. Of course, learning represents a broad activity in its own right encompassing approaches as varied as direct changes in rules to improve performance and the automatic acquisition of knowledge. The former represents a relatively simple approach to automated learning, while the latter is a goal that has yet to be realized.

Most learning, automated or not, operates with some form of feedback. When the feedback comes from a supervisor or teacher, we call this supervised learning. On the other hand, when the feedback derives from internally formulated criteria, we call this unsupervised learning. We begin our discussion of learning with the simplest forms of supervised learning, progress through the more difficult (at least on machines), and end with a description of unsupervised learning.

The simplest form of learning is rote learning, where information found as the result of previous work is stored for reuse. This stored information can derive from input by the user or from a procedure performed by the machine or both. Rote learning is useful because it saves time and computations. Once we have reached a specific useful state, we do not want to have to repeat the work done to get there.

Learning from advice means taking information and converting it into a more useful internal representation. For example, the advice in the card game twenty-one to hold when you have cards with value 17 or higher could be translated into the rule:

$$\text{IF cards\_total\_value} \geq 17 \text{ THEN action} = \text{hold}$$

Learning from advice systems provides mechanisms for this type of translation. However, these systems must check for the consistency of the rule set. Note that the above would be violated by most experienced players who get a pair of aces.

Parameter adjustment represents another form of learning. Many expert systems have parameters (e.g., certainty factors) that can adjust as information arrives. The formal adjustment of these parameters provides an effective mechanism for performance improvement and has been used in many automatic game playing programs [e.g., see (69)].

Learning by induction is the most widely used approach to formal learning both with and without machines. Induction means generalizing from examples. This process is basic to much of science and human understanding. Formal induction encompasses the entire field of statistics. Additionally, many approaches to global optimization derive from principles of induction. From the earliest research into machine intelligence using neural networks, most of the fundamental problems of interest were in the area of induction. Hence, to understand this important area, we group the approaches into the categories of symbolic learning, statistical methods, optimization, and neural networks.

While there are many examples of symbolic learning systems, one that embodies the general idea is the version space approach employed by Mitchell (70). Version spaces maintain a description of a problem-solving situation that evolves with examples. These examples are both positive examples and near misses. For example, to learn the concept of a patient in cardiac distress, the system is presented with examples with different blood pressures and shoe sizes. These examples enable the system to induce that blood pressure is part of the concept of cardiac distress, while shoe size is irrelevant. One algorithm employed to accomplish this type of learning is the candidate elimination algorithm. This algorithm generalizes from positive training examples, so that the general rule must cover all of these. On the other hand, the algorithm takes negative examples and makes the rule more specific. This ensures that these examples are not covered by the rule.

Another symbolic inductive approach is case-based reasoning. This method stores examples within its knowledge base. As new instances are presented to the system, it conducts a search for the closest example in storage. The system then adapts its behavior by working from the most similar instance in memory. The adaptation can use other types of learning, such as parameter adjustment. Issues in case-based reasoning include the representation of the cases, the measures of similarity between cases, and adaptation mechanisms.

As noted above, the entire field of statistics concerns itself with induction or developing general models from specific data sets. Obviously, we will not attempt to describe this rich field in this short section. Instead, we describe an important approach at the intersection between statistics and artificial intelligence: classification trees. Classification trees formulate rules by looking at a set of examples where each example has both a known classification and a fixed number of attributes. For example, suppose we want our system to learn how to classify loan applicants. For simplicity, consider two classes for this example: those who pay back their debts and those who do not. Our example or training set would contain instances of both classes and for each instance, include attributes that would be available from a loan application. These might include current income and current debts. The classification tree would partition the training set according to values of these attributes. The goal is to form this partition in such a way that it will correctly classify future instances (i.e., examples not included in the training set). Successfully classifying these new cases would indicate proper construction of a classification rule or correct induction.

One of the most successful algorithms for performing this type of induction is the recursive-partitioning algorithm used in the Classification and Regression Trees (CART) approach (71). The recursive-partitioning algorithm operates in a manner similar to the old game of 21 questions. The algorithm considers each attribute and asks whether a partition or division on this attribute would successfully group the instances of the training set into their correct classes. For our loan example, the algorithm would look at the current debt attribute and ask if there exists a value such that all members of one class are below, and all members of the other class are above the chosen value. For most problems, a value like this does not exist on a single attribute. Hence, we must find the best partition of the training (the one with the fewest misclassified instances), and then recursively explore the attributes again for each element of this partition. This approach has had numerous successful applications and is now widely available in statistical software packages.

Many researchers have explored optimization methods as the basis for learning by induction. Perhaps the most pervasive and well-known example of this is the use of genetic algorithms. Genetic algorithms (72) model the optimization and learning processes by analogy to evolution. We describe genetic classifiers, which emphasize the learning rather than the optimization side of the field. Suppose we have a population of rules. We also need a fitness function that shows how well each rule performs in the domain of our training set. Our genetic algorithm performs a biased selection of rules fom our initial population based on their fitness. The rules are paired, and then new rules are formed by randomly selecting portions of each parent rule to go into the offspring rules. In this way, the algorithm generates a new population of rules. The process continues until our performance on the training set reaches an acceptable level.

Since rules work in conjunction with other rules, we need a method to assign fitness to the rule that shows its contribution and not that of other rules in chain. Many procedures have been examined to accomplish this task. Holland describes this approach as a bucket brigade in which the ultimate performance is passed back along the rule chain with each rule getting its fair share of the credit.

The intuitive appeal of this approach has led to a large number of experiments and some applications. In most cases, genetic classifiers have run slower than many of the other approaches to induction.

Neural networks represent one of the first approaches to machine intelligence and induction. While genetic algorithms use an analogy with evolution, neural networks use an analogy to the physical structure of the brain. Instead of a single complex processing unit, neural networks attempt to employ many simpler processors working together cooperatively in a network. Many different varieties of neural networks exist, but the best known and most widely employed are multilayer perceptrons, feedforward, or backpropagation networks (BPN).

BPNs organize processors into three or four layers. The processors in the first or input layer take weighted attribute values as inputs. For our loan example, these processors would take as input the income and debt values for the applicant. The weights are multipliers on the input values. The processors in the remaining layers take as input the weighted output values from the processors in the preceding layers. Again, the weights are multipliers on the input lines. The final or output layer produces the classification or response value for the input instance. The layers of processors between the input and output are called hidden layers. At most, two hidden layers are required to learn an arbitrary function.

Each processor's output is the value of a transfer function. While a variety of transfer functions are possible, the one em-

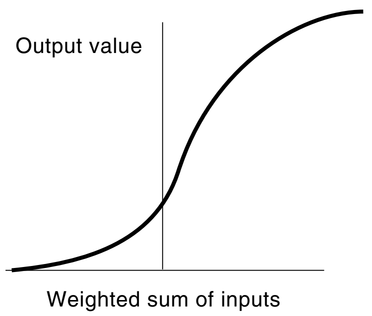

Figure 7. Typical sigmoid function for neural networks.

ployed in BPNs is a sigmoid or s-shaped function (Fig. 7). This particular function guarantees convergence to a local minimum in the error surface that defines the quality of the neural network approximation to the true function.

BPNs learn, as their name implies, through a back propagation algorithm [for details see (73)]. This algorithm takes an instance with a known classification or response value and puts it through the network. The value obtained from the network is then compared to the true value for that class. This error is then propagated back through the network. The algorithm specifies how the weights on each input line to a processor should be adjusted to improve performance and reduce error. The training process is continued until the error is reduced to level and no longer changes significantly between iterations.

BPNs provide an effective approach to learning under a wide variety of problem types. However, they also have many parameters that affect performance. These include parameters in the back propagation algorithm, as well as the topology (number and composition of the layers) of the network. Hence, most development use of BPNs requires considerable experimentation in order to obtain good performance.

The last type of learning is unsupervised learning. In this case, the system develops general rules that organize, group, or cluster members of the training set. For example, we might have a data set provided by the census and want to discover patterns or clusters. As with supervised learning, the instances in the training set have attributes that will serve as the basis for our clustering decisions.

Since we have no supervision, the system does not have examples of correct clusters. Hence, it must use an internal evaluation function to judge how well one particular clustering does in comparison to another. This evaluation function takes as input a measure of similarity or distance between instances and clusters. For census data, the similarity measure would score the similarities between two individuals based on measured attributes of age, address, type of housing, etc. With the evaluation function, clustering algorithms proceed to group the data in a way that puts instances with high similarity together in the same cluster.

Many algorithms exist for performing this function. Unfortunately, the clustering problem itself is among the class of NP-hard problems. Therefore, except for very small problem instances, we cannot obtain a guarantee of optimality for our solutions. Nonetheless, the wide variety of algorithms available can normally provide good solutions for many types of clustering problems encountered in practice.

Knowledge Management and Business Process Reengineering

One of the new phrases in the corporate usage today is the term *knowledge management.* In 1959, management guru Peter Drucker coined the term *knowledge worker* to refer to the day when employees of corporations will be valued more for their cognitive skills and experiences in solving problems rather than their physical (manual labor) skills and experiences (74). Recently, corporate titles such as chief knowledge officer (CKO), chief learning officer, and even chief transformation officer are becoming prominent in major corporations across the country (75).

Knowledge Management (KM) is a topic of growing interest to large organizations. It comprises activities focused on the organization acquiring knowledge from many sources, including its own experience and from that of others, and on the effective application of that knowledge to fulfill the mission of the organization.

As stated in the definition of expert systems, the primary intent of this technology is to realize the integration of human expertise into computer processes. This integration not only helps to preserve the human expertise but also allows humans to be freed from performing the more routine activities that might be associated with interactions with a computer-based system. This makes expert system technology integral to the effective applications of knowledge management in many organizations.

In addition, expert systems have been identified as a key technology in the field of Business Process Reengineering (BPR) (76). BPR is defined as “the fundamental rethinking and radical redesign of business process to achieve dramatic improvements in critical, contemporary measures of performance, such as cost, quality, service, and speed” (76).

Hammer and Champy cite expert systems as a disruptive technology. They cite that sophisticated organizations have learned that “the real value of expert systems technology lies in its allowing relatively unskilled people to operate at nearly the level of highly trained experts” (76). All of this occurs while releasing the experts from their routine problem solving duties to continue to learn and advance in their field and, therefore, become more valuable to the organization.

Distributed Artificial Intelligence and Intelligent Agent Technology

Distributed Artificial Intelligence (DAI) is a rapidly emerging and promising technology. The fundamental objective of DAI technology is to develop “a loosely coupled network of problem solvers—known as a *multi-agent system*—that work together to solve problems beyond their individual capabilities” (77). Expert systems are at the heart of this technology.

There are many key issues in multi-agent systems (MAS) that have yet to be resolved fully. For further discussion of issues in DAI and MAS, see Moulin and Chaib-draa (78). However, a MASS has significant advantages over a single, monolithic, centralized problems solver: faster problem solving by exploiting parallelism, decreased communication by transmitting only high-level partial solutions to other agents rather than raw data to a central site, more flexibility by having agents with different abilities dynamically team up to solve current problems, and increased reliability by allowing agents to take on responsibilities of agents that fail (78).

On-going development of MAS is now progressing, and work has been performed to include expert systems into the mix of problem solvers through cooperating expert systems (79) and as part of larger control systems (80,81).

SUMMARY AND CONCLUSIONS

The purpose of this article has been to present an overview of expert systems technology and its use today and in the future. An attempt has been made to provide the reader with a fundamental understanding of the basic aspects of the technology in terms of its relation to other AI technologies, its structure, its development, and its current and future application.

Expert systems technology is a mature technology and is an integral part of many organizations' decision-making efforts. The practical benefits of the technology have been realized by many organizations, and the future development of these systems will only increase with time due to the fact that more complex problems, in critical domains, can now be addressed.

Future work, however, must still be undertaken in some critical areas, including testing and evaluation of systems and overcoming the limitations of being a brittle, isolated, and static technology.

BIBLIOGRAPHY

1. J. L. Connell and L. B. Shafer, *Structured Rapid Prototyping,* Englewood Cliffs, NJ: Prentice-Hall, 1989.
2. D. E. Brown and J. J. Pomykalski, Reliability estimation during prototyping of knowledge-based systems, *IEEE Trans. Knowl. Data Eng.,* **7**: 378–390, 1995.
3. J. J. Pomykalski and D. E. Brown, Knowledge-based system design enhancement through reliability measurement, *Expert Systems with Applications: An International Journal,* **11** (3): 277–286, 1996.
4. *Merriam-Webster Collegiate Dictionary,* 10th ed., Springfield, MA: Merriam–Webster, 1993.
5. S. J. Rusell and P. Norvig, *Artificial Intelligence: A Modern Approach,* Englewood Cliffs, NJ: Prentice-Hall, 1995.
6. W. S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, *Bull. Math. Biophys.,* **5**: 115–137, 1943.
7. R. K. Lindsay et al., *Applications of Artificial Intelligence for Chemical Inference: The DENDRAL Project,* New York: McGraw-Hill, 1980.
8. E. H. Shortliffe, *Computer-Based Medical Consultations: MYCIN,* New York: Elsevier, 1976.
9. B. G. Buchanan and E. H. Shortliffe (eds.), *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project,* Reading, MA: Addison-Wesley, 1985.
10. J. P. Ignizio, *Introduction to Expert Systems: The Development and Implementation of Rule-Based Expert Systems,* New York: McGraw-Hill, 1991.
11. D. N. Chorafas, *Expert Systems in Manufacturing,* New York: Van Nostrand Reinhold, 1992.
12. J. Durkin, *Expert Systems: Catalog of Applications,* Akron, OH: Intelligent Computer Systems, 1993.
13. D. A. Waterman, *A Guide to Expert Systems,* Reading, MA: Addison-Wesley, 1986.
14. F. Hayes-Roth, D. A. Waterman, and D. B. Lenant (eds.), *Building Expert Systems,* Reading, MA: Addison-Wesley, 1983.
15. P. Harmon and D. King, *Expert Systems: Applications in Business,* New York: Wiley, 1985.
16. J. Durkin, *Expert Systems: Design and Development,* Englewood Cliffs, NJ: Prentice-Hall, 1994.
17. A. J. Gonzalez and D. D. Dankel, *The Engineering of Knowledge-Based Systems: Theory and Practice.* Englewood Cliffs, NJ: Prentice-Hall, 1993.
18. R. J. Mockler and D. G. Dologite, *Knowledge-Based Systems: An Introduction to Expert Systems,* New York: Macmillan, 1992.
19. T. Dean, J. Allen, and Y. Aloimonos, *Artificial Intelligence: Theory and Practice,* Redwood City, CA: Benjamin/Cummings, 1995.
20. G. F. Luger and W. A. Stubblefield, *Artificial Intelligence: Structures and Strategies for Complex Problem Solving,* Reading, MA: Addison-Wesley, 1998.
21. C. H. Cheng, C. W. Holsapple, and A. Lee, Citation-based journal rankings for AI research: A business perspective, *AI Magazine,* **Summer**: 87–97, 1996.
22. F. Hayes-Roth, Rule-based systems, *Commun. ACM,* **28** (9): 921–932, 1985.
23. B. G. Buchanan et al., Constructing an expert system, F. Hayes-Roth, D. A. Waterman, and D. B. Lenat (eds.), *Building Expert Systems,* Reading MA: Addison-Wesley, 1983.
24. D. Partridge, *Artificial Intelligence: Applications in the Future of Software Engineering,* Chichester: Ellis-Horwood, 1986.
25. F. Golshani, Rule-Based Expert Systems, In H. Adeli (ed.), *Knowledge Engineering, Vol. 1: Fundamentals,* New York: McGraw-Hill, 1990.
26. S. M. Weiss and C. A. Kulikowski, *A Practical Guide to Designing Expert Systems,* Totowa, NJ: Rowman & Allanheld, 1984.
27. L. Medsker and J. Liebowitz, *Design and Development of Expert Systems and Neural Networks,* New York: Macmillan, 1994.
28. D. S. Prerau, *Developing and Managing Expert Systems: Proven Techniques for Business and Industry,* Reading, MA: Addison-Wesley, 1990.
29. B. Gaines and M. Shaw, New directions in the analysis and interactive elicitation of personal construct systems, in M. Shaw (ed.), *Recent Advances in Personal Construct Technology,* New York: Academic Press, 1981.
30. B. Gaines, Knowledge acquisition systems, In H. Adeli (ed.), *Knowledge Engineering, Vol. 1: Fundamentals,* New York: McGraw-Hill, 1990.
31. K. Pedersen, Well-structured knowledge bases (Part I), *AI Expert,* **April**: 44–45, 1989.
32. K. Pedersen, Well-structured knowledge bases (Part II), *AI Expert,* **July**: 45–48, 1989.
33. M. Minsky, A framework for representing knowledge, in P. Winston (ed.), *The Psychology of Computer Vision,* New York: McGraw-Hill, 1975.
34. J. L. Kolodner, *Case-Based Reasoning,* San Francisco, CA: Morgan Kaufman, 1993.
35. D. E. Leake, *Case-Based Reasoning: Experiences, Lessons, & Future Directions,* Cambridge, MA: AAAI Press, 1996.
36. B. de Finetti, *Theory of Probability,* Vol. 1, Chichester: Wiley, 1974.
37. B. de Finetti, *Theory of Probability,* Vol. 2, Chichester: Wiley, 1975.
38. V. Barnett, *Comparative Statistical Inference,* Chichester: Wiley, 1982.
39. D. V. Lindley, *Making Decisions,* London: Wiley, 1971.
40. J. Pearl, *Probabilistic Reasoning in Intelligent Systems,* Palo Alto CA: Morgan-Kaufmann, 1988.
41. G. Shafer, *A Mathematical Theory of Evidence,* Princeton, NJ: Princeton University Press, 1976.

42. L. Zadeh, Fuzzy sets, *Information and Control,* **8**: 338–353, 1965.

43. V. L. Yu et al., An evaluation of MYCIN's advice, in B. G. Buchanan and E. H. Shortliffe (eds.), *Rule Based Expert Systems,* Reading, MA: Addison-Wesley, 1984.

44. T. Bahill and M. Jafar, A tool for validating personal computer based expert systems, In C. C. White and D. E. Brown (eds.), *Operations Research and Artificial Intelligence: The Integration of Problem-Solving Strategies,* Baltimore: Operations Research Society of America, 1990.

45. A. Bundy, How to improve the reliability of expert systems, *Seventh Annual Technical Conference of the British Computer Society Specialist Group on Expert Systems,* Brighton, England, 1987.

46. P. N. Finlay, G. J. Forsey, and J. M. Wilson, The validation of expert systems—Contrasts with traditional methods, *J. Operational Res. Soc.,* **39** (10): 2, 1988.

47. L. F. Pau, Prototyping, validation and maintenance of knowledge based systems software, *Proc. IEEE,* 1987.

48. J. T. St. Johanser and R. M. Harbridge, *Validating Expert Systems: Problems and Solutions in Practice,* KBS-86, London, 1986.

49. R. A. Stachowitz and C. L. Chang, *Verification and Validation of Expert Systems,* AAAI-88, St. Paul, MN: MIT Press, 1988.

50. E. Hollnagel, *The Reliability of Expert Systems,* Southampton: Ellis Horwood, 1989.

51. C. J. R. Green and M. M. Keyes, *Verification and Validation of Expert Systems,* WESTEX-87, 1987.

52. R. O'Keefe, O. Balci, and E. P. Smith, Validating expert system performance, In J. S. Chandler and T. P. Liang (eds.), *Developing Expert Systems for Business Applications,* Columbus: Merrill, 1990.

53. D. E. O'Leary, Validation of Expert Systems—With Applications to Auditing and Accounting Expert Systems, *Decision Sciences,* **18**: 1987.

54. D. E. O'Leary, Methods of validating expert systems, *Interfaces,* **18** (6): 1988.

55. A. E. Radwin, et al., A verification approach for knowledge-based systems, *Transportation Research-A,* **23A** (4): 1989.

56. T. Hoppe and P. Meseguer, VVT terminology: A proposal, *IEEE Expert,* **8** (2): 48–55, 1993.

57. J. J. Pomykalski, *Knowledge-Based Systems Design Enhancement through Reliability Measurement,* Ph.D. Dissertation, University of Virginia, 1994.

58. L. Press, Eight-product wrap-up: PC shells, *AI Expert,* **September**: 61–65, 1988.

59. J. Liebowitz, Worldwide perspectives and trends in expert systems: An analysis based on the three world congresses on expert systems, *AI Magazine,* **Summer**: 115–119, 1997.

60. J. Liebowitz and D. S. Prerau (eds.), *Worldwide Intelligent Systems: Approaches to Telecommunications and Networks Management,* Amsterdam: IOS Press, 1995.

61. H. Shrobe, The innovative applications of artificial intelligence conference: Past and future, *AI Magazine,* **Winter**: 15–20, 1996.

62. M. Nahabedian and H. Shrobe, Diagnosing delivery problems in the white house information-distribution systems, *AI Magazine,* **Winter**: 21–30, 1996.

63. D. W. McDonald et al., Desktop underwriter: Fannie Mae's automated mortgage underwriting expert systems, *Ninth Innovative Applications of Artificial Intelligence Conference,* Providence, RI, 1997.

64. F. Highland (guest ed.), Embedded AI, *IEEE Expert,* **9** (2): 18–20, 1994.

65. D. E. Brown and C. C. White (eds.), *Operations Research and Artificial Intelligence: The Integration of Problem Solving Strategies,* Boston: Kluwer, 1990.

66. D. Brown and W. Scherer (eds.), *Intelligent Scheduling Systems,* Boston: Kluwer, 1995.

67. L. R. Medsker, *Hybrid Intelligent Systems,* Boston: Kluwer, 1995.

68. L. R. Medsker, *Hybrid Neural Network and Expert Systems,* Boston: Kluwer, 1994.

69. A. L. Samuel, Some studies in machine learning using the game of checkers, in E. A. Feigenbaum and J. Feldman (eds.), *Computers and Thought,* New York: McGraw-Hill, 1963.

70. T. M. Mitchell, Version spaces: a candidate elimination approach to rule learning, *Proc. Fifth Int. Joint Conf. Artificial Intelligence (IJCAI-77),* 305–310, 1977.

71. L. Breiman et al., *Classification and Regress Trees,* Monterey: Wadsworth, 1984.

72. J. H. Holland, *Adaptation in Natural and Artificial Systems,* Ann Arbor: University of Michigan Press, 1975.

73. P. J. Werbos, Backpropagation through time: What it is and how to do it, *Proc. IEEE,* **78**: 1550–1560, 1990.

74. P. Drucker, *Post-Capitalist Society,* New York: HarperCollins, 1993.

75. D. Bank, Technology: Know-it-alls, *Wall Street Journal,* **28**: 1, November 18, 1996.

76. M. Hammer and J. Champy, *Reengineering the Corporation,* New York: HarperCollins, 1993.

77. E. H. Durfee, V. R. Lesser, and D. D. Corkhill, Trends in cooperative distributed problem solving, *IEEE Trans. Knowl. Data Eng.,* **KOE-11**: 63–83, 1989.

78. B. Moulin and B. Chaib-draa, An overview of distributed artificial intelligence, In G. M. P. O'Hare and N. R. Jennings (eds.), *Foundations of Distributed Artificial Intelligence,* New York: Wiley, 1996.

79. N. R. Jennings et al., Transforming standalone expert systems into a community of cooperating agents, *Engineering Applications of Artificial Intelligence,* **6** (4): 317–331, 1993.

80. D. Cockburn and N. R. Jennings, ARCHON: A distributed artificial intelligence system for industrial applications, In G. M. P. O'Hare and N. R. Jennings (eds.), *Foundations of Distributed Artificial Intelligence,* New York: Wiley, 1996.

81. *LOGOS Requirements and Design Document* (online). Available http://groucho.gsfc.nasa.gov/agents/documents/code522/logos.pdf

JAMES J. POMYKALSKI
James Madison University
WALTER F. TRUSZKOWSKI
NASA GSFC
DONALD E. BROWN
University of Virginia

EXPERT SYSTEMS. See ARTIFICIAL INTELLIGENCE; KNOWLEDGE ENGINEERING; KNOWLEDGE VERIFICATION; MEDICAL EXPERT SYSTEMS.

EXPERT SYSTEMS, AEROSPACE. See AEROSPACE EXPERT SYSTEMS.

EXPLORATION GEOPHYSICS. See GEOPHYSICAL SIGNAL PROCESSING.

EXPOSIMETRY, ULTRASONIC. See ULTRASONIC EXPOSIMETRY.