

FUNDAMENTALS OF EXPERT SYSTEMS

Bruce G. Buchanan

Knowledge Systems Laboratory, Stanford University, Stanford,
California 94305

Reid G. Smith

Schlumberger Palo Alto Research, 3340 Hillview Avenue, Palo Alto,
California 94304

1 INTRODUCTION

Expert systems are among the most exciting computer applications to emerge in the last decade. They allow a computer program to use expertise to assist in a variety of problems, such as diagnosing equipment failures and designing new equipment. Utilizing the results of artificial intelligence (AI) work on problem solving, they have become a commercially successful demonstration of the power of AI techniques. Correspondingly, by testing current AI methods in applied contexts, expert systems provide important feedback to the science about the strengths and limitations of those methods. In this review, we present the fundamental considerations in constructing an expert system, assess the state of the art, and indicate directions for future research. Our discussion focuses on the computer science issues, as opposed to issues of management or application.

1.1 *Characterization and Desiderata*

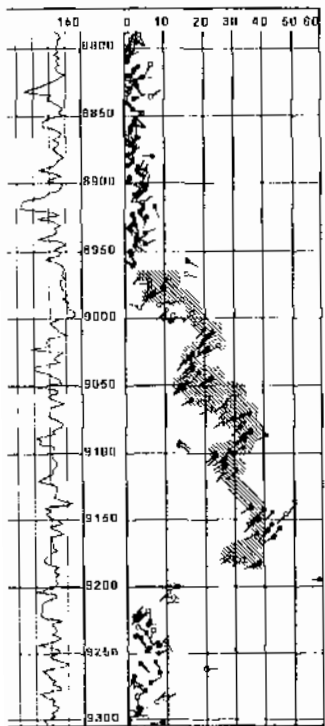
Expert systems are distinguished from conventional programs in several important respects. While none of the characteristics listed below is missing entirely from other well-designed software, all of them together describe a distinct class of programs. Note that few expert systems exhibit all of the following five desiderata to the same degree.

An expert system is a computer program that (a) reasons with domain-

specific knowledge that is symbolic as well as mathematical; (b) uses domain-specific methods that are heuristic (plausible) as well as algorithmic (certain); (c) performs as well as specialists in its problem area; (d) makes understandable both what it knows and the reasons for its answers; and (e) retains flexibility.

An expert system that meets these conditions is the Dipmeter Advisor System (Smith & Young 1984; Smith 1984). Its task is to help petroleum engineers determine the “map” of geological strata through which an oil well is being drilled—e.g. the depth and the dip, or “tilt”, of individual layers of sandstone, shale, and other rocks. It meets our desiderata in the following respects: (a) The knowledge utilized is partly mathematical (e.g. trigonometry) but largely nonnumeric geological knowledge (e.g. about how sand is deposited around river beds). (b) Its reasoning is based on heuristics that well-logging experts use to interpret data from bore holes. (c) It aids specialists, providing interpretations better than those of novices. (d) It uses a variety of graphical and textual displays to make its knowledge understandable and to justify its interpretations. And (e) it is flexible enough to be modified and extended frequently, without rewriting the programs that interpret the knowledge. Figure 1 shows what the Dipmeter Advisor System’s computer screen looks like as an illustration of what a user of an expert system might see.

Characteristics (a) and (b), above—symbolic reasoning and heuristic methods—define expert systems as artificial intelligence programs. Expert systems became an identifiable part of AI in the late 1960s and early 1970s with the realization that application of AI to science, engineering, and medicine could both assist those disciplines and challenge AI. The DENDRAL (Lindsay et al 1980) and MACSYMA (Moses 1971) programs suggested that high performance in a subject area such as organic chemistry was more readily achieved by giving a program substantial subject-specific knowledge than by giving it the general axioms of the subject area plus a powerful, but general, deductive apparatus. The DENDRAL program represented many specific facts about organic chemistry in a variety of ways and used those facts in rather simple inferences. It represented the masses and valences of atoms as values of attributes; it represented classes of unstable chemical compounds as partial graph structures in a table; and it represented certain major patterns of molecular fragmentation in a mass spectrometer as predictive rules. From this work emerged the first principle of expert system building, as enunciated by Feigenbaum (Feigenbaum et al 1971): “In the knowledge lies the power”. The concept of a knowledge base has consequentially become central in expert systems. In contrast, most other AI work of the concerned reasoning by such general methods as theorem proving. Researchers sought to give programs power by means



Type: GROWTH-FAULT
 TOP: 9162.0
 BOTTOM: 9262.0
 Creator: (NORMAL-FAULT-RULESET-0239 . NFR3A)
 Modifier: (.NORMAL-FAULT-RULESET-0239 . NFR3A)
 (NORMAL-FAULT-RULESET-0239 . NFR3A)
 STRIKE: "SSE-NW (159 deg)"
 DIRECTION-TO-DOWNTHROWN-BLOCK: "ENE (68 deg)"
 MIN-FAULT-CUT: 216.0
 ILLUSTRATION:

Normal Fault

rollover

NORMAL-FAULT-0240
 has been asserted by the rule NFR3A
 with TOP = 8950.0
 with BOTTOM = 9262.0
 match variables were :
 :UNCONFORMITY = UNCONFORMITY-0241
 :NORMAL-FAULT = NORMAL-FAULT-0240
 :RED = RED-PATTERN-0197
 :MISSING-SECTION = MISSING-SECTION-0191
 has been modified by the rule NFR3A

NFR3A
 Source:
 J. A. Gilreath
 Author:
 P. Pruchnik & R. Smith, altered by D. Hammock 10-26-84

(In a region where the primary type of distortion is rollover, if there is a normal fault with a red pattern greater than 200 ft. in length associated with it then the fault is probably a growth fault; the fault cuts the well somewhere below the bottom of the red pattern, the strike of the fault is perpendicular to the azimuth of the pattern, the direction to the downthrown block is opposite the azimuth of the pattern & the length of the pattern gives a rough number for the minimum cut of the fault.)

Normal Fault

rollover

Figure 1 Screen from Dipmeter Advisor System: This screen shows the input data and a partial explanation for a conclusion drawn by the system. The left-hand column shows natural gamma radiation against depth. To its right is shown dip against depth. Individual dip estimates (called "tadpoles") show the magnitude of the dip as horizontal position, and the azimuth as a small direction line. High-quality estimates have solid circles, low-quality estimates are hollow. A dip pattern, found by the system, is shown as crosshatching over the relevant tadpoles. To the right are three windows describing the fact that the system has inferred the existence of a Growth Fault, a specialized type of Normal Fault. One window describes the attributes of the fault, another shows a portion of the reasoning trace, and a third describes the rule that made the inference.

of general planning heuristics, exhibited, for example, in problem areas where knowledge about the objects of the domain was almost irrelevant. A favorite problem area was the so-called “Blocks World” of children’s blocks on a table. General knowledge about stability and support, plus general knowledge about planning and constraint satisfaction, allowed programs to reason, say, about the sequence of operations needed to stack blocks in a specified order.

Desideratum *c* separates high-performance programs from others. By specifying human specialists as a standard of comparison, this condition also suggests using the knowledge of specialists to achieve high performance. Predefining the scope of problem solving to a narrow “slice” through a domain (like the slice mastered by most human specialists) has become a pragmatic principle of design. As discussed below, bounding the scope of the problem in advance avoids many of the challenges of building a generally intelligent robot that would behave appropriately in a wide range of situations.

Desiderata *d* (a system’s explaining its reasoning) and *e* (the provision of flexibility) are less frequently cited and less frequently achieved than *a–c*. They may be seen as means of achieving high performance but are included here to highlight their importance in designing and implementing any expert system.

Understandability and flexibility are important both while expert systems are being designed and when they are used. During design and implementation, not all the requisite knowledge is in hand, because not even specialists can say precisely what a program needs to know. Thus expert systems are constructed incrementally. Important to understandability is the use of the same terminology that specialists and practitioners use. Understanding the static knowledge base allows one to decide what knowledge needs to be added to improve performance. Understanding the dynamics of the reasoning is also important in deciding what to change. Flexibility is thus needed to allow the changes to be made easily. Explanations help designers as well as end-users understand the reasons for a program’s conclusions. This capability is especially important when end-users accept legal, moral, or financial responsibility for actions taken on the program’s recommendations.

1.2 Examples

Many expert systems are in routine use (see Rauch-Hindin 1986; Buchanan 1986; Walker & Miller 1986; and Harmon & King 1985 for lists of examples). Some of the best known, such as XCON and the Dipmeter Advisor System, have been used commercially for many years (produced by Digital Equipment and Schlumberger, respectively). The programs shown in Table

I were chosen because they illustrate a variety of problem types and contexts of use. There are roughly two classes of problems addressed in these several systems: (I) problems of interpreting data to analyze a situation, and (II) problems of constructing a solution within specified constraints. Within each category are listed several different examples under general task names that are descriptive but not necessarily distinct.

1.3 *Historical Note*

Early work in AI (1950s–1960s) focused on (a) psychological modeling, and (b) search techniques. Expert systems synthesize some of that work, but shift the focus to representing and using knowledge of specific task areas. Early work used game playing and reasoning about children’s blocks as simple task domains in which to test methods of reasoning. Work on expert systems emphasizes problems of commercial or scientific importance, as defined by persons outside of AI. Newell calls MYCIN “the original expert system” (Foreword to Buchanan & Shortliffe 1984) because it crystallized the design considerations and emphasized the application. Expert systems continue to build on—and contribute to—AI research by testing the strengths of existing methods and helping to define their limitations (Buchanan 1988). In the 1970s expert-systems work developed the use of production systems, based on the early work in psychological modeling. In the 1980s fundamental work on knowledge representation has evolved into useful object-oriented substrates (Stefik & Bobrow 1986).

Hardware developments in the last decade have made a significant difference in the commercialization of expert systems. Stand-alone workstations provide special hardware for AI programming languages, high-resolution interactive graphics, and large address spaces in small boxes at affordable prices (Wah 1987). These have simplified development, since it is no longer necessary to depend on large, time-shared central mainframes for development and debugging. They also provide an acceptable answer to questions of portability for field personnel. Development of expert systems—and the languages and environments (called “shells”) for building them—in standard languages such as Common Lisp and C have essentially eliminated the last barriers to portability.

2 FUNDAMENTAL PRINCIPLES

All AI programs, including expert systems, represent and use knowledge. The conceptual paradigm of problem solving that underlies all of AI is one of search (i.e. a program, or person, can solve a problem by searching among alternative solutions). Although immediately clear and simple, this formulation does not tell us how to search a solution space efficiently and

Table 1 Expert systems working in various problem areas^a**Class I: Problems of Interpretation****Data Interpretation**

- Schlumberger [Dipmeter Advisor System]—interpret down-hole data from oil well bore holes to assist in prospecting (Smith & Young 1984)
- St. Vincents Hospital (Sydney)—aid in interpreting diagnostic tests on thyroid function (Horn et al 1985)
- NL Baroid [MUDMAN]—determine causes of problems in drilling oil wells and recommend additives to the drilling fluid that will correct them (Kahn & McDermott 1986)

Equipment Diagnosis

- General Motors [VIBRATION]—determine causes of vibration noises and recommend repairs (Teknowledge 1987)
- Kodak [BLOW MOLDING INJECTION ADVISOR]—diagnose faults and suggest repairs for plastic injection molding machines (Teknowledge 1987)
- AT&T [ACE]—provide troubleshooting and diagnostic reports on telephone cable problems (Miller et al 1984)
- General Electric [CATS]—diagnose problems in diesel-electric locomotives (Sweet 1985)

Troubleshooting Processes

- Hewlett Packard—diagnose causes of problems in photolithography steps of wafer fabrication (Cline et al 1985)
- Elf Aquitaine Oil Company [DRILLING ADVISOR]—demonstrate reasoning used to find the cause of drill bit sticking in oil wells and to correct the problem (used for training) (Rauch-Hinden 1986)

Monitoring

- IBM [YES/MVS]—monitor and adjust operation of MVS operating system (Rauch-Hindin 1986)
- National Aeronautics and Space Administration [LOX]—monitor data during liquid oxygen tanking process (Kolcum 1986)

Preventive Maintenance

- NCR [ESPM]—monitor computers in the field, analyze error logs, and suggest preventive maintenance procedures before a computer fails (Teknowledge 1987)

Screening

- US Environmental Protection Agency [EDDAS]—determine which requests for information fall under the exceptions to the Freedom of Information Act (Feinstein & Siems 1985)

Credit Authorization

- American Express [AA]—assist in authorizing charges from card members or in determining that a request is suspect or fraudulent (Klahr et al 1987)

Financial Auditing

- Arthur Young [ASQ]—assist auditors with planning and developing approaches to field audits (Hernandez 1987)

Software Consulting

AT&T [REX]—advise persons on which subroutines in large statistical package to use for their problems and how to use them (Rauch-Hinden 1986)

Equipment Tuning

Lawrence Livermore National Laboratory [TQMSTONE]—specify parameter settings to bring a sensitive instrument into alignment (Rauch-Hinden 1986)

Inventory Control

Federal Express [INVENTORY SETUP ADVISOR]—help decide whether or not to stock spares in inventory of 40,000 parts (Teknowledge 1987)

Class II: Problems of Construction

Configuration

Digital Equipment Corp. [XCON]—translate customers' orders for computer systems into shipping orders (Rauch-Hindin 1986)

Design

Xerox [PRIDE]—design paper handling systems inside copiers and duplicators (Mittal et al 1985)

GM Delco Products [MOTOR EXPERT]—generate information necessary to make production drawings for low-voltage DC motor brushes by interacting with designers (Rauch-Hinden 1986)

Loading

US Army (AALPS)—design loading plan of cargo and equipment into aircraft of different types (AALPS 1985)

Planning

Hazeltine [OPGEN]—plan and prepare “operations sheets” of assembly instructions for printed-circuit boards (Rauch-Hindin 1986)

Hughes Aircraft [HI-CLASS]—set up sequence of hand-assembly steps for printed-circuit boards (Hi-Class 1985)

Scheduling

Westinghouse [ISIS]—plan manufacturing steps in Turbine Component Plant to avoid bottlenecks and delays (Fox & Smith 1984)

Babcock & Wilcox—automate generation of weld schedule information (e.g. weld procedure, preheat, postheat, and nondestructive examination requirements) (Rauch-Hindin 1986)

Therapy Management

Stanford Medical Center [ONCOCIN]—assist in managing multi-step chemotherapy for cancer patients (Hickam et al 1985)

^a Although the problems are quite different, they can be categorized into two major classes. We show more than one example of each type to illustrate a range of systems and approaches.

accurately. The number of possible solutions may be astronomical, as illustrated in Table 2. Thus exhaustive consideration of alternatives is out of the question. Most expert systems, however, use heuristics to avoid exhaustive search, much as experts do. For problem areas in which experts are acknowledged to be more efficient and accurate than nonspecialists, it is reasonable to assume that what the experts know can be codified for use by a program. This is one of the fundamental assumptions of knowledge engineering, the art of building expert systems by eliciting knowledge from experts (Hayes-Roth et al 1983).

In this section, we discuss several dimensions of current architectures: representation of knowledge, reasoning, knowledge acquisition, explanation, system-building tools, and validation. In each of these subsections, we try to elucidate the fundamental principles underlying the architectural choices. In the discussion we relate each of the classes of choices to desiderata *a-e* for expert systems, enumerated in Section 1. Several observations others have made about expert systems are restated in the text. Each of these observations is valid for today's expert systems and thus serves to clarify the present state of the art. Underneath each, however, is a trade-off that designers of systems have had to work with. Each observation involves a trade-off. Section 3 discusses research aimed at gaining more of the best of both sides of such trade-offs. We end the section with three advantages of knowledge-based systems over traditional software.

2.1 Representation

One of the hallmarks of an expert system is the use of specific knowledge of its domain of application, applied by a relatively simple inference engine. The phrase "knowledge programming" has been used to denote the emphasis of the effort of building an expert system. The single most important representational principle is the principle of declarative knowl-

Table 2 The size of the solution spaces for several expert systems^a

MYCIN: combinations of 1–6 organisms from list of 120 organisms (many of which are equivalent)	$> 6 \times 10^6$
INTERNIST: combinations of 1–3 diseases from list of 571	$> 31 \times 10^6$
Dipmeter Advisor System: combinations of 650 geological categories for an arbitrary number of depth intervals—e.g. 500 ten-foot intervals	$> (500)^{650}$
XCON: arbitrary number of computer system components selected from 20,000 catalog items 50–150 at a time	$> 10^{200}$

^aThe number of possible solutions defined by the vocabulary of each system is reduced to one, or a few, plausible answers by exploiting domain-specific heuristics.

edge enunciated by McCarthy in the formative years of AI (McCarthy 1958); see also Winograd's discussion of this principle (Winograd 1975). Simply put, this principle states that knowledge must be encoded in an intelligent program explicitly, in a manner that allows other programs to reason about it. Arbitrary Fortran or Lisp procedures, for example, cannot be explained or edited by other programs (although they can be compiled and executed), while stylized attribute-value pairs, record structures, or other, more complex, stylized data structures can be.

To a certain extent, a knowledge base is a database. The essential differences between knowledge bases and databases are flexibility and complexity of the relations. Current research on AI and databases, sometimes called expert database systems (Kerschberg 1986), is reducing these differences. A knowledge base requires an organizational paradigm plus data structures for implementation. These two parts together constitute the representation of knowledge in an AI program.

Elements of knowledge needed for problem solving may be organized around either the primary objects (or concepts) of a problem area or around the actions (including inferential relations) among those objects. For example, in medicine one may think primarily about the evidential links among manifestations and diseases, and the links among diseases and therapeutic actions, and secondarily about the concepts so linked. In this paradigm, one concentrates on the knowledge that allows inferences to be drawn and actions to be taken—the “how to” knowledge. Alternatively, one might organize medical knowledge primarily around the taxonomy of diseases and the taxonomy of their manifestations and secondarily around the inference rules that relate manifestations to diseases. In this second paradigm, one concentrates on what might be called the “what is” knowledge. These two conceptual views are known as “action-centered” or “object-centered” paradigms for representing knowledge. They have counterparts at the implementation level in program organization.

For each type of representation, one may identify the primitive unit and the primitive action. The primitive unit, in the case of action-centered representations, is the fact (e.g. the freezing temperature of water is 0°C). Primitive facts are linked in conditional sentences by rules “if . . . then . . .” statements). Note that these links may reflect causal associations, based on theory, or empirical associations, based on experience. An example from the Dipmeter Advisor System, which is an abbreviated causal description as found in geology texts, is shown in Figure 2.

Conversely, the primitive unit of an object-centered representation is the object, with a number of attributes (called “slots”) and values (e.g. a spur gear with number-of-teeth = 24, material = cast-steel, and diam-

Crevasse-Fan-Rule
<p>if:</p> <p>(1) There exists an element from Sand-Zones in well *Well* <s> (2) There exists an element from Energy-Zones in well *Well* <e>, such that there is an intersection of s and e <i1>, and such that the Energy of e is Moderate (3) There exists an element from Texture-Zones in well *Well* <t>, such that there is an intersection of i1 and t <i2>, and such that the Grain-Size of t is (Fine-Sand Medium-Sand), and such that the Sorting of t is Moderately-Well-Sorted</p> <p>then:</p> <p>(1) Create a Crevasse-Fan-Zone from the top of i2 to the bottom of i2 in well *Well*</p>

Figure 2 Dipmeter Advisor System rule: One of a set used to perform sedimentary environment analysis. This rule is only attempted after the system has determined that the overall sedimentary environment is a deltaic plain.

eter = 5 cm). Objects typically also encapsulate procedures (called “methods”). In addition, they may contain defaults, uncertainty, relations to other objects (e.g. generalizations, parts), and a variety of other information. An object can be viewed as a structured collection of facts. Minsky (1975) popularized the use of objects (then called “frames”) for AI. An example of an object definition from the Dipmeter Advisor System is shown in Figure 3.

Smalltalk (Goldberg & Robson 1983) was one of the early languages that showed both the power of objects as programming constructs and the power of an integrated graphical programming environment. Many commercial expert-system shells contain an object-oriented component (Stefik & Bobrow 1986).

The primitive action in action-centered representations is often referred to as “firing a rule”: If the premise conditions of a conditional rule are true in a situation, then take the actions specified in the consequent part of the rule. For example, in a medical system, conclude that an organism may be streptococcus if its gram stain is positive. This style of programming began as production systems, made popular by Newell’s work in the 1960s (see Buchanan & Shortliffe 1984, Ch. 2).

Observation: The domain models are not “deep” models (Davis 1987). Expert systems rely more on special-case formulations of relations than on “first principles”. Although a set of general principles such as Maxwell’s equations governs the behavior of a large class of devices, designers of expert systems prefer to codify special cases, exceptions, and empirical associations, as well as some causal associations, in order to put the

Normal Fault

Object: Normal-Fault

Synonyms:


Generalizations: Fault

Groups:

Type: Class

Edited: 13-Dec-87 14:26:25 PST **By:** Schoen

Picture[Bitmap]:



Hanging-Wall-Block{Downthrown-Block} [**Object**]:

Upper-Distortion-Region[**Object**]:

Breccia-Region{Crushed-Zone} [**Object**]:

Fault-Plane[**Object**]:

Lower-Distortion-Region[**Object**]:

Foot-Wall-Block{Upthrown-Block} [**Object**]:

Time-Of-Faulting[**Geologic-Age**]:

Strike[**Azimuth**]:

Slip[**Floatingpointnumber**]:

Fault-Angle{Hade} [**Dipmagnitude**]:

Direction-To-Downthrown-Block[**Azimuth**]:

Throw[**Distance**]:

Draw[**Lisp**]: Drawfault

Instantiate[**Lisp**]: Instantiatefault

Detect[**Rule**]: (Rule-Nfr1 Rule-Nfr3 Rule-Nfr4 Rule-Nfr5 Rule-Nfr7)

Specialize[**Rule**]: (Rule-Nfr6 Rule-Nfr9 Rule-Nfr11 Rule-Nfr12)

Figure 3 Dipmeter Advisor System object: Encapsulates information about normal or tensional geologic faults. Individual attribute (slot) names are shown in boldface (e.g. **Hanging-Wall-Block**). Where used, synonyms for attribute names are enclosed in braces (e.g. {Downthrown-Block}). The “type” of each attribute value is shown in square brackets (e.g. the value of the **Strike** slot is expected to be a datum of type [**Azimuth**]).

general principles in forms that can be applied more quickly and more precisely. As a result, they are unable to fall back on a better theory in some situations.

A related observation: Expert systems are “brittle” (Davis 1987, Lenat et al 1986). Without knowledge of first principles, current expert systems may fail precipitously on a new case that is at the boundary of the system’s competence. The performance of humans is more robust: As we reach the extent of what we know about a problem area, we often can give appropriate answers that are approximately correct, although not very precise—and we know the difference. The standard solution today is to codify

rules that screen out cases that are outside the intended scope in order to further ensure that the system is being used in an appropriate way.

Given that rule-oriented programming often involves making deductions, it has been argued that various forms of logic are well-suited for use in expert systems. Simple systems have used propositional logic, more complex systems have used first-order predicate logic, and there is ongoing research in use of higher-order logics to express relations among beliefs, temporal relations, necessity, and uncertain information (both the uncertainty with which data must be regarded in many real systems and uncertainty about the strength of heuristic rules, which reflects a lack of detailed understanding of a domain).

Observation: The reasoning is not formal (Nilsson 1982). Many designers of expert systems are uncomfortable with mathematical logic as a representation language because it lacks expressive power. Numerous extensions must be made to express some of the concepts that are frequently used in applications: uncertainty, strategy knowledge, and temporal relations. Some logicians are uncomfortable with reasoning that is not theorem proving and with knowledge bases that are not axiomatic systems that allow proofs of consistency and completeness. The search for new logical formalisms that are more powerful than predicate calculus reflects the tension between simple, well-understood formalisms and expressive power.

In object-centered representations, the primitive action is called “sending a message”: If an action needs to be taken (e.g. a value of an attribute is needed), send a request to the object that can take the action (e.g. compute, or conclude, the value). For example, in a geology system, send the *Analyze-Sedimentary-Environment* message to an instance of the *Borehole-Interval* object. The effect is to perform an arbitrary action, which could include drawing inferences; in our example, the action performed is to draw conclusions about the geological “story” of sedimentation at a specific depth interval penetrated by the oil rig’s drill. This style of object-oriented programming was defined by Hewitt (Hewitt 1977).

In terms of data structures, objects are much like record structures. Each object has a number of fixed fields. Unlike record structures, however, new fields can be added to objects during a computation. Objects divide into two types: instances and classes. Instances are individuals in a domain (e.g. a specific depth interval from 1200 to 1225 feet in a specific borehole). Classes represent sets of individuals (e.g. any depth interval). They define the characteristics of the individuals that are their instances. Classes are usually organized into hierarchies according to different relations. The most common relations are the specialization, subclass, or “is-a” relation (e.g. a reverse geological fault is a kind of geological fault) and the “part-of” relation (e.g. a fault plane is part of a geological fault). Object-oriented

systems allow arbitrary relations to be encoded, but often provide efficient support for one or two specific relations.

In order to support the characteristics of expert systems listed in the Introduction, representation mechanisms must have sufficient expressive power to state, clearly and succinctly, both “what is” knowledge and “how to” knowledge. [This is sometimes called the “epistemological adequacy” of a representation (McCarthy & Hayes 1969)]. Expressive power has both design-time and run-time implications. One of the key problems for designers of expert systems is management of complexity. Impoverished representation mechanisms force designers to encode information in obscure ways, which eventually leads to difficulty in extending and explaining the behavior of expert systems. Representation mechanisms that permit efficient compilation and structuring of knowledge reduce run-time requirements of both time and memory.

As an example, an object-oriented language allows some information to be stated once, in an abstract class, and accessed (by inheritance) in a large number of subclasses. A representational mechanism that does not allow this forces designers to confront the complexity of stating essentially the same information many times. This may lead to inconsistency and difficulty in updating the information. It also has an obvious memory cost. At run time, each of the separate encodings of the information may have to be considered individually, resulting in an obvious performance penalty. An example of a taxonomic hierarchy is shown in Figure 4.

Action-centered and object-centered paradigms are in fact two ends of a spectrum of representational possibilities. The two emphasize different aspects of modeling. Contemporary AI programs often use heterogeneous representational paradigms (e.g. coupling the simplicity of rules with the expressive power of objects).

Extensible representation schemes facilitate the incremental development of expert systems, which is necessary when there is no complete specification of either the problem or the knowledge required to solve it. When new concepts, attributes, and relations are added incrementally, a designer must not be forced to recode substantial portions of the knowledge already encoded.

Observation: Knowledge bases are not reusable (Lenat et al 1986). Since the cost of building a knowledge base is substantial, it is desirable to amortize it over several related expert systems, with unique extensions to cover unique circumstances. For example, many medical systems use facts about anatomy and physiology, yet often each encodes those facts specifically for use in a unique way. The challenge is to develop knowledge representations that can be used efficiently, independent of the specific context of use.

Experience has shown that declarative, modular representations are useful for expert systems. Some information is more difficult to encode in

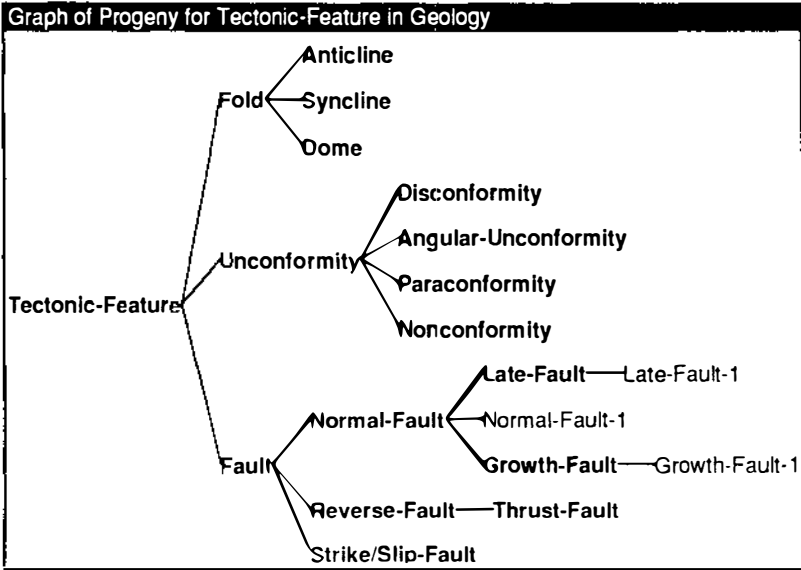


Figure 4 Dipmeter Advisor System Tectonic Feature hierarchy: Subclasses of each object are shown in boldface, to its right, connected to it by lines. Individual instances are shown in lightface.

the action-centered paradigm, other information more difficult in the object-centered paradigm. For example, sequencing of actions is difficult to encode in an action-centered paradigm. The same is true of information that is essentially static, such as causal or structural descriptions. On the other hand, object-centered representations have no built-in inference mechanism beyond inheritance (although they support them, and many commercial shells have an integrated rule-oriented component). In addition, in some domains, subclasses are “soft” and it may be inappropriate to wire in hard distinctions between classes (e.g. in geology, classification of rocks according to lithology—sandstone, shale, carbonate—is not firm because the end-members are mixed to varying degrees). Consequently, there is no single answer to the question “Which representation method is best?” Contemporary expert systems use a variety of methods but attempt to integrate them into a uniform framework. As systems become more complex, it will be more and more difficult to maintain a uniform view.

2.2 Reasoning

Inference methods are required to make appropriate and efficient use of the items in a knowledge base to achieve some purpose, such as diagnosing

a disease. Logically speaking, the two rules of inference most used in problem solving are *modus ponens* (“If A implies B and you know A, then infer B”) and *modus tollens* (“If A implies B and you know not-B, then infer not-A”). The former is sometimes called the “chain rule” because inferences can be chained together in a sequence of deductions:

$$\begin{array}{l} A \\ A \rightarrow B \\ B \rightarrow C \\ C \rightarrow D \\ \hline \end{array}$$

Therefore, D

In addition to these two simple rules, rules of quantification are sometimes used—e.g. “If all As are Bs and x is an A, then x is a B”. With a few simple rules of inference such as these driving the problem solving, a knowledge base full of many special facts and relations about the problem area can provide the expertise on which high performance is based.

Observation: Expert systems make little use of common sense reasoning (McCarthy 1983). Designers of current expert systems resolve this by (a) assuming that users can exercise some common sense, and (b) specifying common facts explicitly when needed. The INTERNIST system, for example, contains about 100,000 commonsense medical facts such as “males do not get pregnant” and “aspirin obscures the results of thyroid tests” (R. Miller, personal communication). The challenge is to construct a “commonsense reasoning component” that is general enough to avoid errors that “any fool” would avoid and specific enough to reason reliably and efficiently.

Some expert systems (e.g. those written in Prolog) use a theorem prover to determine the truth or falsity of propositions and to bind variables so as to make propositions true. Others use their own interpreters in order to incorporate more than a theorem prover provides—most importantly, capabilities for controlling the order of inferences, strategic reasoning, and reasoning under uncertainty. Most fielded rule-based expert systems have used specialized rule interpreters, not based directly on logic. To some extent this reflects timing—efficient Prolog interpreters and compilers have only recently become available (Clocksin & Mellish 1981). However, it also reflects a need for more flexible styles of inference (in addition to a theorem prover’s depth-first backtracking) and control over the strategies guiding the order of inferences.

2.2.1 CONTROLLING THE ORDER OF INFERENCES AND QUESTIONS From a logical point of view, the order in which new facts are derived is irrelevant, if all logical consequences of the initial facts are to be considered. For pragmatic reasons, expert systems often need to be selective about which

facts to consider and which consequences to pursue. Space and time are often limited, for example, and it may also be important to develop a line of reasoning that a user can follow. Thus, expert systems are organized around three different reasoning paradigms: forward, backward, and opportunistic reasoning.

Forward reasoning from data to conclusions is used when the cost or inconvenience of gathering data is low and there are relatively few hypotheses to explore. A forward-chaining system starts with a collection of facts and draws allowable conclusions, adding those to the collection and cycling through the rules. The stopping conditions vary from stopping with the first plausible hypothesis to stopping only when no more new conclusions can be drawn. The XCON computer configuration system is a classic example of a forward-chaining system.

Expert systems may be faced with inconsistent or time-varying data. As a result, the reasoning employed is often nonmonotonic—i.e. conclusions may be altered or withdrawn as problem solving proceeds. This, too, necessitates a departure from a traditional logical view (Bobrow 1980).

Matching the premise clauses of all rules in a knowledge base against each new situation can be prohibitively expensive when there are many rules and many new situations created by inferring new facts. Rules often contain variables that can be bound in many different ways, thus creating additional ways that their premises can match a situation. Rule interpreters commonly provide mechanisms for the compilation of rules and rule-matching procedures (Brownston et al 1985). In addition, all but the simplest rule-based systems organize and index rules in groups in order to control the expense of matching and invocation. Rule groups (called “rule sets”, “tasks”, or “control blocks”) are also used to control the expert system’s focus of attention in order to make interactions with users more comprehensible. For example, in a medical system, it helps users understand the reasoning if data requests are clustered by grouping rules that (a) perform disease diagnosis, (b) focus on the patient’s history or on the laboratory tests, and (c) recommend therapy. (This also facilitates the acquisition of knowledge and the maintenance of knowledge bases.)

Backward reasoning is goal-directed and does not require all relevant data to be available at the time inferences are begun. It is more appropriate when a user supplies many of the data, and when the user cares about the order in which data are requested. MYCIN is a classic example. A backward-chaining system starts with a hypothesis (goal) to establish and asks, in effect, “What facts (premise clauses of rules) would need to be true in order to know that the hypothesis is true?” Some of these facts may be known because they were given as initial data, others may be known after asking the user about them, and still others may be known only after

starting with them as new subgoals, and chaining backward. The stopping conditions vary from stopping with the first hypothesis found true (or “true enough”) to stopping only after all possibly relevant hypotheses have been explored.

Opportunistic reasoning combines some elements of both data-directed (forward) and goal-directed (backward) reasoning. It is useful when the number of possible inferences is very large, no single line of reasoning is likely to succeed, and the reasoning system must be responsive to new data’s becoming known. As new data are observed, or become known, new inferences can be drawn; and as new conclusions are drawn, new questions about specific data become relevant. An opportunistic reasoning system can thus set up expectations that help discriminate a few data elements from an otherwise confusing mass. The key element of such a system is an agenda of actions with an associated scheduler that enables explicit decisions to be made about which actions are to be taken (e.g. which rules to apply, whether to apply them in a forward- or backward-chaining manner, and which object is to be the focus of attention). Such decisions, by contrast, are hard-wired into forward- and backward-chaining systems. One successful prototype based on this paradigm is the HASP system (Nii et al 1982). Acoustic data from sensors in the ocean provide information about the types and locations of vessels. As data are received over time, hypotheses are revised. With each revision, new ambiguities arise, which can be resolved by reprocessing old data or looking for new signals.

2.2.2 USING EXPLICIT STRATEGIES The three major reasoning paradigms of forward, backward, and opportunistic reasoning are primitive strategies that may need refinement and coordination in order to reflect a complex decision strategy such as medical diagnosis. Representing strategic knowledge explicitly, an important trend in expert systems, becomes important whenever strategic issues are subject to change or explanation. MYCIN’s metarules, a solution to this problem in the late 1970s, represent knowledge of reasoning strategy as rules (Buchanan & Shortliffe 1984). They differ from the other “domain knowledge” rules in the system in that they refer to those rules in some of their premise or conclusion clauses:

IF \langle medical context \rangle AND there are rules that mention fact A and that mention fact B,
THEN reason with the rules mentioning A before the others.

Strategies can also be represented as an organization of steps to perform, in a stylized definition of a procedure (Clancey 1986; Hickam et al 1985).

Explicit representation of strategy knowledge can also improve a systems's ability to explain its own behavior.

2.2.3 REASONING UNDER UNCERTAINTY Reasoning under uncertainty is essential in problem areas outside of logic and mathematics, in which information is incomplete or erroneous. In medicine, for example, there is rarely complete certainty about having *all* the data or about the accuracy of the data. Several methods are used in expert systems to deal with uncertainty arising from (*a*) uncertainty of the data, (*b*) less than certain associations between data and conclusions, and (*c*) combinations of these. The major methods for addressing these issues are listed below.

1. Abstraction—assume that the uncertainty is small and can safely be ignored. The method is extremely simple and efficient to use. However, many problems require more precision in estimating uncertainty.
2. Bayes's Theorem—use prior and posterior probabilities to represent less than certain data and associations; then compute new probabilities with some variation of Bayes's Theorem (Gorry 1970). This method is based on a solid formalism, but it requires either frequency data or subjective estimates for many combinations of events.
3. Fuzzy Logic—represent the uncertainty of propositions such as “John is tall” with a distribution of values; then reason about combinations of distributions (Zadeh 1979). This is intuitively appealing because it is based on ordinary linguistic concepts. It is computationally more complex than other mechanisms because it propagates uncertainty through distributions of values.
4. Criterion Tables—assign categories or weights to clauses in rules based on their relative importance in drawing conclusions (e.g. major and minor findings associated with a disease); then allow a conclusion to be drawn if sufficient numbers of clauses in each category are true (Kulikowski & Weiss 1982). This is a simple mechanism that is computationally very fast. It fails to capture gradations between categories, however, and thus lacks the expressive power to reason in some complex problem areas.
5. Certainty Factors (CFs)—assign single numbers to propositions, and to associations among propositions, representing either probabilities or a combination of probabilities and utilities; then use MYCIN's formulas to determine CFs for inferred beliefs (Buchanan & Shortliffe 1984). This calculus has been frequently used and has been shown to have a formal interpretation in probability theory. Since it is based on measures of increased belief, the effects of adding new relations or changing CFs may be difficult to predict.

A general problem with methods 2–5 is arriving at a coherent set of numbers. Typically these are obtained from experts over several iterations, with empirical testing, because valid, objective numbers are not available.

2.2.4 SUMMARY There is no single answer to the question, “Which inference method is best?” Each expert system, or system-building shell, provides a nearly unique set of choices for controlling the inferences, using strategies and reasoning under uncertainty. Some feature still other issues, such as methods for backtracking (recovering from local failures), critiquing (making no recommendations unless the user needs them), reasoning about shapes or positions, and reasoning about temporal dependencies. Most present-day systems allow no modification of the inference methods they use. This is a shortcoming that has not received widespread attention, but that causes system builders to make inappropriate or unhappy choices because they must work with an inference procedure within a shell in which someone else made those choices.

2.3 *Knowledge Base Development*

For the last decade, everyone involved has referred to the process of putting knowledge into a knowledge base as a “bottleneck” in building expert systems (Hayes-Roth et al 1983). Usually this process involves two persons (or teams): an expert whose knowledge is to be partially mirrored in the knowledge base, and a knowledge engineer who interviews the expert to map his/her knowledge into the program’s data structures. The process is time-consuming and difficult, yet the performance of the resulting expert system depends on its being done well. This is exacerbated by the fact that knowledge base design often involves integrating the knowledge of several experts, because relying on a single expert may cause implicit assumptions to be overlooked. A survey conducted by SRI International indicates that the average cost of developing an application (knowledge engineering plus end-user interface alone) is about \$260,000. For small systems, these costs are about \$5000; for large systems, more than \$1.5 million (Fried 1987). Note that these estimates do not include the cost of constructing an expert system shell.

Much of the process of knowledge engineering is engineering. Yet there are several different issues of a fundamental nature wrapped up in the steps of the process.

1. During the first step, problem assessment, the knowledge engineer must match characteristics of the proposed problem against characteristics of known solution methods. Unfortunately there are no good taxonomies

of either problems or solution methods and no good criteria for deciding that there is a match.

2. The second major step is exploratory programming, in which a series of experimental prototypes are constructed quickly, first as proof-of-concept, then with successively larger fractions of an expert's knowledge, showing that a part of the problem can be (partially) solved with that knowledge encoded in a specific environment. Two substantial issues here are (a) formulating an accurate conceptual framework, including terminology, to allow knowledge to be added incrementally; and (b) interacting with—not just passively listening to—the expert efficiently to elicit what he/she knows about the problem that is relevant for the expert system.

3. Developing the knowledge base, to increase both the breadth and depth of the system's competence, is the third major step. This step takes the most time (several person-years) but is relatively straightforward if steps 1 and 2 have been done well. One difficult issue here is anticipating characteristics of end-users and their context of use. Another is deciding which new facts and relations are and which are not relevant for the system's performance and understandability in context. The competing paradigms for making this decision—and for knowledge engineering generally—may be called “model-directed” and “case-directed” knowledge base development. In the former, the knowledge base is largely developed along the lines of a model, or theory, of the problem area. In the latter, it is largely developed in response to errors exhibited in solving test cases. Neither is entirely adequate by itself; knowledge engineers must use both. Whatever combination of development paradigms is used, there is no clear stopping criterion for development. This presents problems in providing for continual additions and modifications to a knowledge base—the extensibility mentioned above.

4. The last step of the process is software engineering, to ensure that the system fits into the end-users' environment, is responsive to their needs, etc. The difficult issues at this step are not unique to expert systems. It is included as a reminder that a successful application requires more than developing a knowledge base.

Observation: Expert systems do not learn from experience (Schank 1983). Research on machine learning is maturing to the point where expert systems will be able to learn from their mistakes and successes. Learning by induction from a large library of solved cases is already well enough understood to allow induction systems to learn classification rules that an expert system then uses (Michie et al 1984; Michalski et al 1986). Prototype systems have been built that emphasize learning in context, sometimes called explanation-based learning or apprentice learning, which appears to hold promise for expert systems (Mitchell et al 1986). The challenge is to design learning mechanisms that are as accurate as knowledge engineering but are more cost effective.

2.4 *Explanation*

One of the defining criteria of expert systems is their ability to “explain” their operation. Early forms of explanation focussed on showing the line of reasoning, typically a sequence of rule firings, that led to a particular conclusion. This was normally done in stylized natural language (Buchanan & Shortliffe 1984, Pt. 6). The user could ask the system questions of the form “How did you conclude. . . .” In a sense it is an extension to the kind of dialog that was originally shown in the SHRDLU system (Winograd 1972). That system answered questions by actually looking in its environment and on its own goal stack (i.e. agenda of goals and subgoals).

Although natural language interfaces were used almost exclusively in early expert systems, powerful, low-cost graphics workstations have fueled a trend towards graphical interfaces [e.g. the STEAMER system, used to train naval personnel to operate steam power plants onboard ships (Hollan et al 1984)]. Contemporary systems often provide mixed natural language and graphical interfaces [e.g. the Drilling Advisor System (Rauch-Hindin 1986)].

Lines of reasoning [e.g. the GUIDON-WATCH System (Richer & Clancey 1985)] may be shown as graphs that permit user interaction to explore alternative possible lines of reasoning. Perhaps this makes clear the fact that current explanation facilities are much like sophisticated program debugging facilities and are often used as such. Like all good debugging systems, they permit the programmer/user to examine system operation in high-level terms, rather than in terms of the low-level machine instructions actually executed. There is a trend today towards recording justifications that underlie the items in the knowledge base (Smith et al 1985). These can be used to augment explanations. Research is ongoing to enable expert systems themselves to use this information.

The term “explanation” can also be used to cover examination of the static knowledge base. Object-oriented representations and sophisticated graphics facilities enhance the ability of a domain specialist to understand what has been encoded (Smith et al 1987). As found in the GUIDON system (Clancey 1986), however, such facilities do not in and of themselves constitute a tutoring system.

Observation: The expert’s conceptual framework may not be the same as the users’ (Winograd & Flores 1986). Knowledge engineers work under the assumption that the experts they work with know the context of intended use and the intended users’ terminology and point of view. This may result in misuse of a system when a user attaches different meanings to terms than did the expert who designed the knowledge base. There are no safeguards built into today’s systems to test this assumption. Thus the challenge is to provide enough ways of explaining what is in the knowledge base to make its contents clear to all users.

A related observation: Expert systems have little self-knowledge (Lenat et al 1983). While expert systems can often give explanations of what they know, they do not have a general “awareness” of what the scope and limitations of their own knowledge are. Metalevel knowledge, such as rules of strategy, can offset this shortcoming in special situations but does not constitute a general capability.

One could argue that the user of a conventional Fortran program can also examine the “knowledge base” of the program. Depending on how the program is written, this is true to a certain extent. It would typically be done with a text editor. One thing that sets expert systems apart, however, is their ability to be queried in the run-time context. Whereas a conventional program can be examined only statically, an expert system can be examined dynamically. It is true that a programmer can examine the stack of a conventional program with a debugger, but such programs do not maintain an explicit goal stack or line of reasoning. This is not a statement about implementation language but rather about system design style.

2.5 *System-Building Tools/Shells*

When the first commercial expert systems were being developed, the developers were faced with two major problems: (a) eliciting and encoding the domain knowledge necessary to solve the problem at hand, and (b) building programming systems with which to encode/apply the knowledge. There were almost no generally applicable rule interpreters or object-oriented programming languages. Most of the early “shells” had been constructed in universities as parts of specific applications. They typically made too many assumptions about either the domain of application of the problem-solving methods to be used. Furthermore, they were typically only usable by highly trained specialists. Finally, their run-time, space, and implementation language requirements precluded their use in a wide variety of environments. Nevertheless, these shells represented generalizations, in code, of principles learned from experience with prior expert systems.

One of the most practical effects of the recent commercial application of expert systems is the development of many dozens of robust shells and tool sets (Bundy 1986; Gevarter 1987; Harmon 1987; Richer 1986). These shells range in capability from those that can support little more than experimentation with rule-based techniques, to those that can support efficient development and operation of substantial systems. A few of the more powerful shells are used to support current research in expert systems. The shells are implemented in a number of programming languages (e.g. Lisp, C, Prolog) and run on a variety of hardware, including inexpensive PCs, workstations and mainframe computers.

Today, users can expect a high-end shell to offer support for a number

of programming paradigms. The two most common are rule-oriented programming and object-oriented programming. Both forward and backward chaining are standard, as is support for structuring rules into collections (or rule sets) according to task. Rules are typically efficiently compiled into code in the underlying implementation language. Not all rule languages are extensible. The OPS5 rule language, for example, allows new action functions to be defined but does not allow new matching predicates (Brownston et al 1985).

When support for object-oriented programming is provided, it includes multiple inheritance, message-passing, and active values. A common way to combine rules and objects is to construct a method that responds to a message by applying a set of rules, with either forward or backward chaining. Such a method may also be invoked in response to a change in an active value. The REACTORS system, for example, uses active values to respond to changes in the operating conditions of a nuclear power plant to invoke rules that suggest new responses (Rauch-Hindin 1986).

Some shells provide support for uncertainty in rules and in facts. The certainty factor calculus originally developed for the MYCIN system is widely used. Complete integration of inexact reasoning and objects has not yet been achieved. It is currently limited to support of uncertainty for slot values. Support for uncertainty in inter-object relations is less common.

In the early years of commercial systems, expert systems were designed as stand-alone tools. As a result they were not well integrated with database management systems, large numerical packages, or other existing software and systems. Today's commercial systems are considerably better integrated with other uses of computers. It is now common to see support for mixed-language environments (e.g. with some code in Lisp and some in C).

Over the past few years, increasing attention has been focused on tools to support interaction between humans and expert systems. There are two major reasons for this: (a) in many fielded systems the end-user interface accounts for a substantial portion of the overall system, and success depends heavily on the quality of user interaction (Smith 1984); and (b) the knowledge-acquisition process is simplified and enhanced when the expert can readily examine the evolving knowledge base and directly interact with the system to refine its understanding of the domain (e.g. Davis & Lenat 1982). It has also been found that the tools used to represent domain knowledge and strategy knowledge (e.g. objects and rules) can be applied to structuring user interfaces. Extensible systems and tools have been developed to support interaction requirements for knowledge engineers, experts, and end-users (Smith et al 1987).

2.6 *Validation*

There are many dimensions along which we might wish to judge an expert system. The three most important of these are computational, psychological “look and feel,” and performance. Computational issues include speed, memory required, extensibility, and portability. Psychological issues include ease of use, understandability and “naturalness,” and online help. Performance issues—the sine qua non—include the scope of competence, percentage of false positive and negative solutions (false hits and misses), and time or money saved. Some involve evaluations of the static knowledge base (e.g. its scope) while others involve looking at the program in use (e.g. its ease of use or statistics on correctness).

Formal validations of expert systems are rarely published, if done at all. The formal validation of MYCIN’s performance (Buchanan & Shortliffe 1984, Pt. 10) stands out as an exception. In that study, outside evaluators reviewed therapy recommendations, for several randomly selected patients, as made by MYCIN and nine persons whose expertise ranged from acknowledged specialist to medical student. The evaluators (in a blinded study) judged MYCIN’s recommendations to be indistinguishable from those of the specialists. In practice, expert systems are validated in the same way as conventional software. Developers demonstrate that a new system solves a variety of difficult problems before it is turned over to end-users (O’Keefe et al 1987). A few of the end-users then try the new system in context on a large number of cases, often in parallel with the old method for solving these problems. Any errors that are detected are fixed. When the end-users and their managers are convinced of the program’s effectiveness, the program is put into routine use, often at a single site first.

With conventional programs, we often test each branch of each subroutine with boundary values of variables, to assure ourselves that the program’s parts behave as specified. In an expert system, each element of the knowledge base is examinable in the same fashion as a single, small subroutine. As with subroutines, the places where unforeseen errors occur are in the interactions among the elements. These have to be uncovered by empirical tests—running the program on a large, random sample of problems (within the specified scope) and determining which cases are solved correctly and which not. In the absence of a complete logical analysis that proves the correctness of both the knowledge base and the inference engine, we must analyze performance empirically. The criteria for “acceptable” levels of errors of any type, however, must be determined by weighing costs of errors of each type against the benefits of correct solutions.

2.7 *Advantages over Traditional Software*

In general, the main issues in building expert systems revolve around complexity, interpretability, and explicit modular forms of knowledge. In this section we summarize some of the advantages of using expert systems instead of writing conventional software.

2.7.1 COMPLEXITY

Complexity of problem Often when one begins designing an expert system, neither the problem nor the knowledge required to solve it is precisely specified. Initial descriptions of the problem are oversimplified, so the complexity becomes known only as early versions of the system solve simple versions of the problem. Expert systems are said to approach competence incrementally. A declarative, modular representation of knowledge, applied in a uniform manner, is the key to managing this kind of complexity.

Complexity of project management The traditional life-cycle model of software construction and maintenance presumes that problems are specified. An alternative model, used in constructing expert systems, is exploratory programming in which problem definition and problem solution are mutually reinforcing. A key element in exploratory programming is a powerful, integrated development environment (Sheil 1984).

Complexity of system Conventional software can in principle be written by good programmers to solve any problem that an expert system solves. Frequently a system that is initially constructed in a shell system is rewritten in Fortran, PLI, C, or some other well-known language. Constructing the system in the first place, however, requires considerably more flexibility than is provided in a non-interpreted language, unless the designer has considerably more ability than most, or unless the shell system (itself in C or some other language) provides an interpreter for elements in its knowledge base.

2.7.2 INTERPRETATION One of the facilities commonly used to advantage in expert systems is evaluation—*EVAL* to the Lisp programmer. This facility allows the user (or the system itself) to specify a query or arbitrary computation to the running system and evaluate it in the run-time context. It lays open to examination the entire state of the system and its environment, including the knowledge base, the line of reasoning, agenda, etc. This is the sense in which programs written in interpretive languages like Lisp are said to themselves constitute data. It is one of the most important facilities upon which an expert system depends. It allows a system to reason not only about incoming data but also about past inferences and even about how it makes inferences. To a certain extent, operating systems

also perform this kind of introspection. However, these systems can usually only be tuned in a number of pre-defined ways, according to a fixed set of parameters; operating systems typically cannot look at their own procedures. By contrast, expert systems in principle can do this kind of detailed introspection, examining their procedures as well as their data.

In order for this capability to be effectively used, it is important that the knowledge be represented explicitly (declaratively) and uniformly, and that it be applied in a relatively uniform manner. While it may be possible in principle to reason about pure Lisp code, in practice it is extremely difficult—for humans as well as programs.

2.7.3 KNOWLEDGE Specialized knowledge of a problem area is the key to high performance. And the key insight from AI has been that representing a program's knowledge declaratively provides considerable advantages over hard-wiring what a program knows in coded subroutines. There is a continuum, of course, from parameterized procedures to completely stylized, understandable, high-level procedure descriptions; and today's expert systems have room to improve. As discussed extensively above, the central knowledge issues in building expert systems are: representation, reasoning, acquisition, and explanation. Today's expert systems demonstrate the adequacy of current AI methods in these four areas, for some well-chosen problems. Shells, or system-building environments, codify many of the present methods. Yet there remain limitations on what can be easily represented, used, acquired, or explained.

3 STATE OF THE ART

Several recent books and publications provide extensive overviews and details about the state of the art. See, for example, Waterman (1986), Rauch-Hinden (1986), Mishkoff (1985), and Scown (1985) plus numerous current journals and newsletters such as *Expert Systems*, *IEEE Expert*, *The AI Magazine*, *Expert System Strategies*, and *The Applied Artificial Intelligence Reporter*. In this section we encapsulate our own understanding of the state of the art.

3.1 *Size of System*

The numbers of expert systems and persons working on them have grown to the point where building expert systems has become routine. While their size and scope are definitely limited, it is difficult to characterize them, either numerically or symbolically. For example, MYCIN contained about 1,000 rules and 20 class names, and XCON contains about 6,000 rules and 100 class names. The INTERNIST system contains about 2,600 rules,

with another 50,000 links among roughly 600 diseases (objects), and 80 manifestations (slots) per disease (chosen from approximately 4,500 manifestations in all). Numbers like these are difficult to compare because (a) there may be substantial differences in the level of conceptual detail covered in a rule in different shells (e.g. EMYCIN vs OPS5); (b) there is more in a knowledge base than rules and object names; (c) complex procedures contain considerable knowledge, even though not represented declaratively; and (d) a single concept, or a single clause in a rule, may stand for something very complex (e.g. “state of the patient”) or for something quite straightforward (e.g. “patient’s age”). As developers attempt to encode more information in objects (attempting to make fewer assumptions about how the knowledge will be used), the number of rules tends to be reduced in a faster than linear fashion. This occurs because the rules are written to be applied to members of hierarchically organized classes of objects, and not just to single individuals.

A few expert system shells have small upper limits on the size of the knowledge base that can be accommodated, mostly for reasons of memory size of the underlying personal computer. Even systems that today are counted as modestly large or complex mention only a few thousand objects (or classes of objects) and relations among them (e.g. rules). These limits may be due to experts’ and knowledge engineers’ limitations in keeping track of larger numbers of items (and their interactions)—and to managers’ unwillingness to spend more than 12–24 months in developing a system—and not to hardware or software limits. New technology will be required, however, when we try to build knowledge bases that contain millions of items. An approximate characterization of the complexity of present-day knowledge bases is shown in Table 3. Assuming that facts are represented as object-attribute-value triples (e.g. “the identity of Organism-2 is *E. coli*”), it makes some sense to ask how many there are. There are complications, however, because (a) classes may be defined for arbitrarily many instances and (b) attributes may take on continuous values (e.g. any real number). So instead of showing the number of facts, Table 3 shows the number of components of facts. Also, instead of showing only the number of rules, this table indicates the depth and breadth of inference networks. It also suggests that knowledge bases are more complex when they must deal with uncertain facts and relations.

The time it takes to build a system varies greatly depending on the scope of the problem and the expectations about the end product. A prototype that is expected to demonstrate feasibility on a small troubleshooting problem, for example, may be built by a single person in one to ten weeks. A fully developed system ready for field use on a complex problem, on the other hand, may take a team of several persons one to three years or more.

Table 3 Approximate measures of complexity of expert systems built routinely in the late 1980s^a

Vocabulary	
Number of objects	1000s of objects or classes of objects
Number of attributes per object	10–250 named attributes
Number of legal values per attribute	3–100 discrete values, or arbitrarily many discrete ranges of values of continuous attributes
Inferential Relations (Rules)	
Number of rules	100s to 1,000s
Depth of longest chains	2–10 steps from primary data to final conclusion
Breadth of inferences	2–10 ways of inferring values of any single attribute
Degrees of uncertainty	facts and relations may be expressed with degrees of uncertainty

^a These numbers represent empirical, not theoretical, upper bounds on several key parameters.

One measure of our increased understanding of knowledge programming is that students are now routinely assigned one-term class projects that would have been two-year doctoral research projects a decade ago.

3.2 *Type of Problem*

Several types of problem for which systems can be built were listed above in two categories: interpretation and construction. We lack a robust taxonomy of problem types (among the best so far is the one proposed by Chandrasekaran 1986), so the individual examples still provide a better characterization of the types of problem than general descriptions. Most expert systems described in the open literature address problems of data interpretation, mostly for purposes of troubleshooting or equipment diagnosis. They are mainly organized around the method of evidence gathering, in which evidence is gathered for and against a fixed set of hypotheses (or solutions), and the answer(s) with the best evidence is selected (Buchanan & Shortliffe 1984). This is also known as catalog selection or heuristic classification (Clancey 1985). Most of the commercial shells address problems of this type. However, more and more systems are being built for problems of the second category, and shell systems are emerging to handle the myriad constraints that shape a design, assembly, configuration, schedule, or plan.

Observation: Expert systems do not reason exactly as human experts do—e.g. they have no intuition (Dreyfus & Dreyfus 1986). So far, the problems that have been most successfully solved with expert systems have been those in which inferential knowledge

is easily formulated as rules and the organization of objects and concepts is easily formulated as taxonomic (class-subclass-instance) hierarchies and part-whole hierarchies. Reasoning by analogy or by intuition is still too unpredictable (and ill-understood) to use in high-performance systems.

3.3 *Some Limitations and Research Topics*

Expert systems are designed to solve specific problems in well-circumscribed task domains in which specialists can articulate the knowledge needed for high performance. Current methods for designing and building them have limitations, briefly discussed as observations above. These limitations intersect somewhat with the research issues shown in Table 4. The difference between them is one of emphasizing performance (pragmatics) or issues (theory). In each of these areas some work has been done. To date, however, proposed methods have not been well integrated with shell systems, often because proposed methods have not been convincingly generalized or demonstrated. Table 4 is thus a partial list of doctoral dissertation topics brought into focus by work on expert systems. General solutions to any of these problems would constitute valuable contributions to AI.

The limitations mentioned briefly above, although stated negatively to indicate the boundaries of what is common practice, also indicate directions in which expert systems research is growing. Some partial solutions to some of these shortcomings have been elucidated in research laboratories and a few are exhibited in commercial systems. We will see more of these capabilities integrated in fielded systems of the future.

4 CONCLUSIONS AND SUMMARY

4.1 *Design Principles*

Out of the experimental work with expert systems over the last five to ten years, several “architectural principles” of expert systems have emerged. In 1982, Davis articulated an early set of principles based on experience with a few rule-based systems (Davis 1982; see McDermott 1983 for another set of generalizations, and Hayes-Roth et al 1983, Ch. 5, for practical advice for knowledge engineers). Given additional experience, we can augment and refine these principles.

4.1.1 MODULAR, DECLARATIVE EXPRESSIONS OF KNOWLEDGE ARE NECESSARY

1. *Represent all knowledge explicitly.* This simplifies explanation of system behavior as well as refinement, both by human designers and by the system itself. The main feature of an expert system is the suite of specific knowledge it has about its domain of application. For reasons

Table 4 Nineteen research topics in expert systems^aTopics in Representation

Structure & Function—how to represent structural and functional models and their interdependencies (e.g. the way primitive physical parts, such as integrated circuit packages and wires, are placed and connected to make up the physical structure of a device, as opposed to the way the composition of the logical functions implemented by the primitive parts leads to the behavior embodied by the device). Progress in this area could lead to more robust diagnostic systems. They will be able to deal with faults whose diagnosis depends on knowledge of the interplay between the structural and functional decompositions of a device.

Continuous Space & Time—how to represent (and reason efficiently about) arbitrary spatial regions and intervals of time. Specialized knowledge of spatial and temporal reasoning processes is essential for systems that deal with geometry (e.g. mechanical CAD/CAM systems) and time (e.g. budget planning systems). In a sense this is “domain knowledge” that is applicable to a variety of domains.

Processes—how to represent explicitly knowledge about processes and procedures. Again, specialized knowledge is required for expert systems in areas like semiconductor or chemical manufacturing.

Problem-Solving Methods—how to represent knowledge for solving specific classes of problems (e.g. heuristic classification, constraint propagation, top-down refinement, means-ends analysis). Expert systems shells can be made more powerful by inclusion of “generic” knowledge in particular areas (as noted above for Continuous Space & Time or Processes). They can also be made more powerful by inclusion of knowledge of particular problem-solving methods.

Reusability—how to represent domain knowledge or strategy knowledge so that it can be used in different applications and extended by different users. The challenge is to encode knowledge in such a way as to be general enough to apply in a number of contexts, yet specific enough to provide real power.

Topics in Reasoning

Scale—how to store, access, and reason efficiently with knowledge bases that are orders of magnitude larger than today’s (millions of items instead of thousands). This is essentially an engineering problem. Progress will involve techniques for reducing storage, techniques for dealing with knowledge bases that cannot be entirely loaded into virtual memory, improved knowledge indexing techniques for faster access, and techniques that allow distributed access and update.

Interactions—how to reason effectively about multiple, interacting problems (e.g. faults in a device whose effects reinforce or mask each other or that otherwise provide test results different from the union of results for individual faults).

Integration—how to exploit the special-purpose reasoning methods in existing software packages (e.g. spreadsheets) with knowledge-based reasoning.

Distributed Knowledge—how to pass information (data, problems, and solutions) and coordinate activity in a network of distributed problem solvers (machine and human) reliably and efficiently. Progress in this area will lead to effective support for “virtual team” design and engineering.

Parallel Problem Solving—how to solve parts of a problem simultaneously on different computers and synthesize a solution.

First Principles—how to represent and use theoretical laws of prediction for other purposes (such as design or failure diagnosis), and how to effectively combine this type of reasoning with the use of simple associations. Progress will lead to systems that combine the robustness of first-principle and causal reasoning with the efficiency of associational reasoning.

Constraint-Based Reasoning—how to efficiently represent and reason with dependencies among problem parameters. Progress is expected in languages for expressing constraints, and in techniques for reasoning based on constraint propagation, explicit solution of possibly nonlinear equations, and optimization. Ways of representing the degree to which constraints must be satisfied and control strategies able to selectively relax constraints are required. New techniques are also required to permit effective man-machine interaction for constraint-based reasoning systems.

Analogical Reasoning—how to find “reasonable” pairs of problems or knowledge bases that make useful analogies, and how to use all and only “relevant” mappings out of the thousands of possible mappings.

Topics in Knowledge Acquisition

Intelligent Editors and Debugging Tools—how to assist in designing and building an expert system for a specialized task without already possessing knowledge about that task area.

Learning—how to learn new knowledge from present experience, or from libraries of past problems; how to learn from data that may be erroneous. A central issue will be learning in the context of routine use, with feedback from users, as in the apprentice learning model.

Consistency—how to find inconsistencies in a knowledge base, especially when it contains items with degrees of uncertainty; and how to suggest ways of making groups of items consistent.

Multiple Sources of Knowledge—how to combine the contributions of many different specialists into a coherent knowledge base, especially when their knowledge is seemingly contradictory or is framed in incompatible vocabularies.

Topics in Explanation

Customized Explanations—how to tailor an explanation for an individual use and context without pre-specifying answers for each different class of situations. Important components of progress in this area will be methods for representing and reasoning about human users, their goals, and the purposes for which they desire explanations. This includes providing intelligent summaries of a knowledge base or a line of reasoning that takes into account the background and assumptions of the person requesting the summary. It also includes explaining a process as well as explaining the contents of the knowledge base.

Topics in Shells

Knowledge Compilation—how to compile rules and objects efficiently while preserving explanation capabilities.

^a Each topic represents an open problem whose solution will enhance both the understanding of artificial intelligence and the performance of expert systems.

of extensibility and flexibility, it is important to separate the abstract concepts and relations of the target domain from inferences that can be made in the domain—i.e. “what is known” from “how to use it.”

2. *Keep elements of the knowledge base as independent and modular as possible.* When updating rules or links among objects, the fewer the interactions with other parts of the knowledge base the easier the isolation and repair of problems. Although complete independence of rules or objects is impossible (without complex, lengthy descriptions of the context of relevance), partitioning the knowledge base into small, nearly independent modules facilitates maintenance. Common partitionings include: (a) domain-specific knowledge (e.g. a model of structural geology, which could be used in a variety of applications), (b) task-specific knowledge (e.g. the knowledge of how to use the model of structural geology, together with a model of the data sensed by a dipmeter tool, to interpret the data in terms of geological structures), (c) knowledge about interaction with developers and users, (d) problem-solving knowledge (e.g. strategies like top-down refinement and least-commitment constraint propagation), and (e) other domain-independent knowledge (e.g. commonsense facts, mathematics, etc).
3. *Separate the knowledge base from the programs that interpret it.* Historically this has been phrased as “separate the knowledge base and the inference engine” (Davis 1982).
4. *Consider interaction with users as an integrated component.* It is important to avoid dealing with user interaction issues in an “add on” manner, after the expert system has been designed. High-quality user-interaction frameworks are often essential to end-user utility. They are also important to easing the knowledge-acquisition bottleneck.
5. *Avoid assumptions about context of use.* Extending a knowledge base is made difficult when assumptions about how the individual packets of knowledge will be used are implicitly encoded. For example, important premise conditions of a rule may be omitted because the system developer knows the context in which that rule will be applied (as noted earlier with the sample rule from the Dipmeter Advisor System). This is also important if domain-specific knowledge bases are to be reused for a variety of applications.

4.1.2 UNIFORMITY, SIMPLICITY, EFFICIENCY, AND EXPRESSIVE POWER ARE INTERDEPENDENT

1. *Use as uniform a representation as possible,* although specialized representations are often worth the cost of translating among represen-

tations, because they may improve run-time performance and simplify knowledge acquisition.

2. *Keep the inference engine simple.* A program's ability to reason about its actions depends on its ability to reason about the way it makes inferences, and complex inference procedures make this task more difficult. But this may cause problems in expressing knowledge in "appropriate" ways and in run-time efficiency.
3. There is a logical equivalence among representational choices, but *an object-centered paradigm offers the most flexibility*, and thus the most expressive power.
4. *Be sure the reasoning is based on sound, conceptually simple strategic knowledge.* A knowledge base is more than a bag of facts and relations; it is used for a purpose with a reasoning strategy in mind. The clearer that strategy is, the more coherent the knowledge base will be. However, this may reduce run-time performance.

4.1.3 REDUNDANCY IS DESIRABLE

1. *Exploit redundancy.* One advantage of a modular representation of the domain knowledge is that it allows the system to explore multiple lines of reasoning. By contrast, a conventional program typically has a single procedure with a fixed sequence of steps for achieving a goal. Reasoning with uncertain or missing data, or with knowledge that is uncertain or incomplete, requires building redundancy into the reasoning to allow correct conclusions to be drawn in spite of these deficiencies.

4.2 Summary

Expert systems use AI methods for representing and using experts' knowledge about specific problem areas. They have been successfully used in many decision-making contexts in which (a) experts can articulate much of what they know (e.g. in training manuals), (b) experts reason qualitatively (e.g. based on what they have learned from experience) to augment the formulas in textbooks, and (c) the amount of knowledge required to solve problems is circumscribed and relatively small.

While there remain many open research problems of great interest and importance, expert systems—and the shell systems that are generalizations of them—encapsulate solutions to many problems associated with the representation, use, acquisition, and explanation of knowledge. The engineering solutions used in today's expert systems are not without limits, but they are well-enough understood and robust enough to support commercial applications. Moreover, each application provides more experimental data about the strengths of current AI methods.

ACKNOWLEDGMENTS

This work was supported in part by DARPA (Contract N00039-86-C-0033), NIH (Grant RR-00785), NASA (Contract NCC 2-220-S1), and by Schlumberger.

Eric Schoen and David Hammock assisted in generating the Dipmeter Advisor System figures. Discussions with Robert Young helped in formulating architectural principles of expert systems.

Literature Cited

- AALPS. 1985. SRI: AI and the Military. *Artif. Intell. Rep.* 2(1): 6-7
- Bobrow, D. G., ed. 1980. *Artificial Intelligence. Special Issue on Non-Monotonic Logic*, 13: 1-1-2. New York: North-Holland
- Brownston, L., Farrel, R., Kant, E., Martin, N. 1985. *Programming Expert Systems in OPS5*. Reading, Mass: Addison-Wesley
- Buchanan, B. G. 1986. Expert systems: working systems and the research literature. *Expert Syst.* 3(1): 32-51
- Buchanan, B. G. Artificial intelligence as an experimental science. In *Aspects of Artificial Intelligence*, J. H. Fetzer, ed. Amsterdam: D. Reidel
- Buchanan, B. G., Shortliffe, E. H. 1984. *Rule-Based Expert Systems: the MYCIN Experiments of the Stanford Heuristic Programming Project*. Reading, Mass: Addison-Wesley
- Bundy, A., ed. 1986. *Catalogue of Artificial Intelligence Tools*. New York: Springer-Verlag, 2nd ed.
- Chandrasekaran, B. 1986. Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design. *IEEE Expert* 3: 23-30
- Clancey, W. J. 1985. Heuristic classification. *Artif. Intell.* 27: 289-350
- Clancey, W. J. 1986. From GUIDON to NEOMYCIN and HERACLES in twenty short lessons: ONR final report 1979-1985. *AI Mag.* 7(3): 40-60, 187
- Cline, T., Fong, W., Rosenberg, S. (1985). *An expert advisor for photolithography*. Tech. Rep., Hewlett-Packard, 1501 Page Mill Rd., Palo Alto, CA 94304
- Clocksink, W. F., Mellish, C. S. 1981. *Programming in Prolog*. New York: Springer-Verlag
- Davis, R. 1982. Expert systems: where are we? and where do we go from here? *AI Mag.* 3(2): 1-22
- Davis, R. 1987. Robustness and transparency in intelligent systems. In *Human Factors in Automated and Robotic Space Systems*, pp. 211-33. Washington, DC: Committee on Human Factors, Natl. Res. Council
- Davis, R., Lenat, D. B. 1982. *Knowledge-Based Systems in Artificial Intelligence*. New York: McGraw-Hill
- Dreyfus, H., Dreyfus S. 1986. Why expert systems do not exhibit expertise. *IEEE Expert* 1(2): 86-90
- Feigenbaum, E. A., Buchanan, B. G., Lederberg, J. 1971. On generality and problem solving: a case study using the DENDRAL program. In *Machine Intelligence 6*, ed. B. Meltzer, D. Michie, pp. 165-90. New York: American Elsevier
- Feinstein, J. L., Siems, F. 1985. EDAAS: An expert system at the US Environmental Protection Agency for avoiding disclosure of confidential business information. *Expert Syst.* 2(2): 72-85
- Fox, M. S., Smith, S. F. 1984. ISIS—A knowledge-based system for factory scheduling. *Expert Syst.* 1(1): 25-49.
- Fried, L. 1987. The dangers of dabbling in expert systems. *Computerworld* 21: 6 ff.
- Gevarter, W. B. 1987. The nature and evaluation of commercial expert system building tools. *Computer* 20(5): 24-41
- Goldberg, A., Robson, D. 1983. *Smalltalk-80: The language and its Implementation*. Menlo Park: Addison-Wesley
- Gorry, G. A. 1970. Modelling the diagnostic process. *J. Med. Educ.* 45: 293-302
- Harmon, P. 1987. Currently available expert systems-building tools. *Expert Syst. Strat.* 3(6): 11-18
- Harmon, P., King, D. 1985. *Expert Systems: Artificial Intelligence in Business*. New York: John Wiley
- Hayes-Roth, F., Waterman, D. A., Lenat, D. B., eds. 1983. *Building Expert Systems*. Reading, Mass: Addison-Wesley
- Heckerman, D. E. 1986. Probabilistic interpretations for MYCIN's certainty

- factors. In *Uncertainty in Artificial Intelligence*. NY: North Holland
- Hernandez, R. 1987. Big eight firm audits with Mac. *Appl. Artif. Intell. Rep.* 4(7): 9
- Hewitt, C. 1977. Viewing control structures as patterns of passing messages. *Artif. Intell.* 8: 323-64
- Hi-Class. 1985. AI brings smarts to PC-board assembly. *Electronics* 58: 17-18
- Hickam, D. H., Shortliffe, E. H., Bischoff, M. B., Scott, A. C., Jacobs, C. D. 1985. The treatment advice of a computer-based cancer chemotherapy protocol advisor. *Ann. Intern. Med.* 103: 928-36
- Hollan, J. D., Hutchins, E. L., Weitzman, L. 1984. STEAMER: an interactive inspectable simulation-based training system. *AI Mag.* 5(2): 15-27
- Horn, K. A., Compton, P., Lazarus, L., Quinlan, J. R. 1985. An expert computer system for the interpretation of thyroid assays in a clinical laboratory. *Aust. Comput. J.* 17(1): 7-11
- Kahn, G., McDermott, J. 1986. The Mud system. *IEEE Expert* 1(1): 23-32
- Kerschberg, L., ed. 1986. *Expert Database Systems: Proceedings of the First International Workshop*. Menlo Park: Benjamin Cummings
- Klahr, P., et al. 1987. The authorizer's assistant: a large financial expert system application. In *Proceedings of the Third Australian Conference on Applications of Expert Systems*, pp. 11-32. Sydney: New South Wales Inst. Technol.
- Kolcum, E. H. 1986. NASA demonstrates use of AI with expert monitoring system. *Aviation Week & Space Technol.*, March, pp. 79-85
- Kulikowski, C., Weiss, S. 1982. Representation of expert knowledge for consultation: The CASNET and EXPERT projects. In *Artificial Intelligence in Medicine*, ed. P. Szolovits, pp 21-55. Boulder, Colo: Westview Press
- Lenat, D. B., Prakash, M., Shepherd, M. 1986. CYC: Using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks. *AI Mag.* 5(4): 65-85
- Lenat, D. B., Davis, R., Doyle, J., Genesereth, M., Goldstein, I., Schrobe, H. 1983. Reasoning about reasoning. In *Building Expert Systems*, ed. F. Hayes-Roth, D. A. Waterman, D. B. Lenat. Reading, Mass: Addison-Wesley
- Lindsay, R. K., Buchanan, B. G., Feigenbaum, E. A., Lederberg, J. 1980. *Applications of Artificial Intelligence for Organic Chemistry: The DENDRAL Project*. New York: McGraw-Hill
- McCarthy, J. 1958. Programs with common sense. In *Proc. Symp. Mechanisation of Thought Processes*, pp. 77-84, Nat. Phys. Lab. [Reprinted 1968 in *Semantic Information Processing*, ed. M. L. Minsky, pp. 403-9. Cambridge, Mass: MIT Press
- McCarthy, J. 1983. Some expert systems need common sense. *Ann. NY Acad. Sci.* 426: 129-37. (Invited presentation for the NY Acad. Sci. Sci. Week Symp. Computer Culture, April 5-8)
- McCarthy, J., Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*, ed. B. Meltzer, D. Michie, pp. 463-502. Edinburgh: Edinburgh Univ. Press
- McDermott, J. 1983. Extracting knowledge from expert systems. In *IJCAI-83*, 1: 100-7. Karlsruhe, West Germany: IJCAI
- Michalski, R. S., Moztetic, I., Hong, J., Lavrac, N. 1986. The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proc. AAAI-86*, pp. 1041-45. Philadelphia, Penn: AAAI
- Michie, D., Muggleton, S., Raster, C., Zubrick, S. 1984. RULEMASTER: a second-generation knowledge-engineering facility. In *The First Conference on Artificial Intelligence Applications*, pp. 591-97. Silver Spring, Md: IEEE, IEEE Computer Society Press
- Miller, F. D., Copp, D. H., Vesonder, G. T., Zielinski, J. E. 1985. The ace experiment: initial evaluation of an expert system for preventive maintenance. In *Artif. Intell. Maintenance: Proc. Joint Serv. Workshop*, ed. J. Richardson, pp. 421-27. Park Ridge, NJ: Noyes
- Minsky, M. 1975. A framework for representing knowledge. In *The Psychology of Computer Vision*, ed. Patrick H. Winston, pp 211-77. New York: McGraw-Hill
- Mishkoff, H. C. 1985. *Understanding Artificial Intelligence*. Dallas, Tex: Texas Instruments Information Publishing Center 75265
- Mitchell, J. M., Carbonell, J. G., Michalski, R. S., eds. 1986. *Machine Learning: A Guide to Current Research*. Boston: Kluwer Academic Publications
- Mittal, S., Dym, C. L., Morjaria, M. 1985. PRIDE: an expert system for the design of paper handling systems. In *Applications of Knowledge-Based Systems to Engineering Analysis and Design*, ed. C. L. Dym. NY: ASME Press
- Moses, J. 1971. Symbolic integration: the stormy decade. *Commun. ACM* 8: 548-60
- Nii, H. P., Feigenbaum, E. A., Anton, J. J., Rockmore, A. J. 1982. Signal-to-symbol transformation: HASP/SIAP case study. *AI Mag.* 3(2): 23-35
- Nilsson, N. J. 1982. *Symbolic Computation:*

- Principles of Artificial Intelligence*. Berlin: Springer-Verlag
- O'Keefe, R. M., Balci, O., Smith, E. P. 1987. Validating expert system performance. *IEEE Expert* 2(4): 81-89
- Rauch-Hindin, W. B. 1986. *Artificial Intelligence In Business, Science, and Industry: Volume I—Fundamentals; Volume II—Applications*. Englewood Cliffs, New Jersey: Prentice-Hall
- Richer, M. H., Clancey, W. J. 1985. Guidonwatch: a graphic interface for viewing a knowledge-based system. *IEEE Comput. Graph. Appl.* 5(11): 51-64
- Richer, M. H. 1986. Evaluating the existing tools for developing knowledge-based systems. *Expert Syst.* 3(3): 166-83
- Schank, R. C. 1983. The current state of AI: one man's opinion. *AI Mag.* 4(1): 3-8
- Scown, S. J. 1985. *The Artificial Intelligence Experience*. Maynard Mass: Digital Equipment Corp. 183 pp.
- Sheil, B. A. 1984. Power tools for programmers. In *Interactive Programming Environments*, ed. D. R. Barstow, H. E. Shrobe, E. Sandewall, New York: McGraw-Hill, p. 19-30
- Smith, R. G. On the development of commercial expert systems. *AI Mag.* 5(3): 61-73
- Smith, R. G., Young, R. L. 1984. The design of the dipmeter advisor system. *Proc. ACM Annu. Conf. ACM, New York, October* pp.15-23
- Smith, R. G., Winston, H. A., Mitchell, T. M., Buchanan, B. G. Representation and use of explicit justifications for knowledge base refinement. In *Proceedings of IJCAI85*, pp. 673-80. Los Altos, Calif: Morgan Kaufmann
- Smith, R. G., Barth, P. S., Young, R. L. 1987. A substrate for object-oriented interface design. In *Research Directions in Object-Oriented Programming*, ed. B. Shriver, P. Wegner, pp. 253-315. Cambridge, Mass: MIT Press
- Stefik, M. J., Bobrow, D. G. 1986. Object-oriented programming: themes and variations. *AI Mag.* 6(4): 40-62
- Sweet, L. 1985. Research in progress at General Electric. *AI Mag.* 6(3): 220-27
- Teknowledge. 1987. *TEKSolutions: Customer Success Stories*. Teknowledge, 1850 Embarcadero Road, Palo Alto, Calif., 94303
- Wah, B. W., ed. 1987. *IEEE Computer (Spec. Iss. Comput. AI Appl.)* 20(1)
- Walker, T. C., Miller, R. K. 1986. *Expert Systems 1986*. Madison, Ga: SEAI Tech. Publ.
- Waterman, D. A. 1986. *A Guide to Expert Systems*. Reading, Mass: Addison-Wesley
- Winograd, T. 1972. *Understanding Natural Language*. New York: Academic
- Winograd, T. 1975. Frame representations and the procedural/declarative controversy. In *Representation and Understanding: Studies in Cognitive Science*, ed. D. G. Bobrow, A. Collins, pp. 185-210. New York: Academic
- Winograd, T., Flores, F. F. 1986. *Understanding Computers and Cognition*. Norwood, NJ: Ablex
- Zadeh, L. A. 1979. A theory of approximate reasoning. In *Machine Intelligence 9*, ed. J. E. Hayes, D. Michie, L. I. Mikulich, pp. 149-95. Chichester: Ellis Horwood Ltd.