AI Magazine Volume 26 Number 4 (2006) (© AAAI)

IAAI/AI Magazine *2006 Robert Engelmore Award Address*

What Do We Know about Knowledge?

Bruce G. Buchanan

■ Intelligent systems need knowledge. However, the simple equation "knowledge is power" leaves three major questions unanswered. First, what do we mean by "knowledge"; second, what do we mean by "power"; and third, what do we mean by "is"? In this article, I will examine the first of these questions. In particular I will focus on some of the milestones in understanding the nature of knowledge and some of what we have learned from 50 years of AI research. The discipline and detail required to write programs that use knowledge have given us some valuable lessons for implementing the knowledge principle, one of which is to make our programs as flexible as we can.

Thank you for this distinguished award and the opportunity to share some thoughts with you.[1] What I would like to give you in this article[2] are some of the principles guiding the implementation of knowledge-based systems that follow from work in philosophy and AI. Many of them are well known, but they can serve as reminders of the difficulty of implementing the "knowledge is power" principle.[3] I wish to clarify the knowledge principle and try to increase our understanding of what programmers and program designers need to do to make the knowledge principle work in practice.

The "knowledge is power" principle is most closely associated with Francis Bacon, from his 1597 tract on heresies: "Nam et ipsa scientia potestas est." ("In and of itself, knowledge is power.") Incidentally, Bacon was probably as much interested in the political power to be gained from knowledge as the power to understand nature,[4] but the concept of knowledge is much the same.

Bacon was among the first of the modern philosophers to separate the concept of scientific knowledge from knowledge gained through the two dominant methods for attaining truth in his time: magic and religious revelation. The essential difference for him, as for us, is that knowledge gained through experiment is replicable by others.

Although all the empirical sciences rely on the replication of observations and experiments, AI has been slow to embrace this principle.[5] Programs demonstrating research ideas in AI are often too large and not well enough documented to allow replication or sharing. Applications programs, however, are designed to be used by others outside the research lab and thus are more amenable to multiple runs in diverse conditions. Thus they have the potential to provide experimental data demonstrating strengths, weaknesses, and benefits.

Contributions before AI

The knowledge principle predates Bacon. For example, it was pretty clearly articulated in Biblical times: "A man of knowledge increaseth strength" (Proverbs 24: 5).

Socrates, Plato, Aristotle, and other early Greek philosophers based their lives on acquiring and transferring knowledge. In the course of teaching, they sought to understand the nature of knowledge and how we can establish knowledge of the natural world.

Socrates is famous for pointing out the value of knowledge and seeking truth, as in "… that which we desire to have, and to impart to oth-

 ISSN 0738-4602

ers, [is] expert knowledge…." (Plato, Phaedrus 270d).

He was also fond of pointing out how little we actually know—and was put to death, essentially, for pointing that out to everyone:

> When I conversed with him I came to see that, though a great many persons, and most of all he himself, thought that he was wise, yet he was not wise. Then I tried to prove to him that he was not wise, though he fancied that he was. … I thought to myself, "I am wiser than this man: neither of us knows anything that is really worth knowing, but he thinks that he has knowledge when he has not, while I, having no knowledge, do not think that I have. I seem, at any rate, to be a little wiser than he is on this point: I do not think that I know what I do not know." Next I went to another man… (Plato, *Apology* VI:22).

Plato, Socrates's pupil and Aristotle's mentor, was the first to pose the question in writing of what we mean when we say that a person knows something about the world (Cornford 1935). He was distinguishing empirical knowledge, lacking complete certainty, from the certain knowledge of mathematics. The whole dialogue, *The Theaetetus*, is worth reading but—if I may oversimplify the conclusion—Plato, speaking for Socrates, concludes that person *S* knows empirical proposition *p* if and only if:

> *S* believes *p*
> *p* is true (otherwise it is a false belief, not a fact that is known)
> *S* can provide a rationale for why *p* is true (which Plato calls giving an account).

The last condition has been modified by philosophers in recent years to read that *S* is justified in believing *p*. This modification preserves the requirement of a rationale but removes the onus of providing it from subject *S*. That is, the belief is justified, but *S* does not need to be the one providing it. But, of course, philosophers are not at all in agreement about what constitutes a proper justification or rationale. One view that would seem to be relevant for AI is that either the belief or the justification is formed through a reliable cognitive process. *S* didn't just come to believe *p* through a series of bad inferences or guessing based on the wrong reasons (Steup 2006).

Aristotle continued the search for knowledge, extending the methodology in two important ways beyond the rational discussion of Plato and the mathematics of Pythagoras. His term for science, incidentally, was "natural philosophy," which was used by scientists as late as Newton to describe their own work. One of Aristotle's most lasting contributions was showing the importance of knowledge gained through observation, as opposed to pure reason. Aristotle wrote at least 31 treatises describing every aspect of the natural world and offering physical explanations of many phenomena.

Aristotle also advanced the rational tradition of Plato and Pythagoras by developing a logic that captures many forms of symbolic argument, which was powerful enough to survive two thousand years. He demonstrated the expressive power of simple propositions, "*A* is *B*," along with quantification, "All *A*'s are *B*'s," or "Some *A*'s are *B*'s." He also established rules of symbolic inference for combining quantified propositions.

Euclid's geometry firmly established the concept of rigorous proof within mathematics. Some of the Greek philosophers' contributions to our concept of knowledge are highlighted in table 1.

In the intervening several centuries before the Middle Ages and the rise of modern science in the West,[6] the search for knowledge was overwhelmed by the power of the Christian church to make new knowledge fit with established dogma. The resulting dark ages should be a reminder to all of us that knowledge-based systems should not merely perpetuate the established dogma of an organization.

The English theologian and philosopher Robert Grosseteste (1170–1253) is known for emphasizing the role of mathematics in understanding the natural world. Galileo later underscored this principle when he wrote that the "book of nature" is written in "the language of mathematics." Grosseteste is also credited with establishing the experimental method as a path to knowledge in his own experimental work on the refraction of light.

William of Ockham was the most influential philosopher of the 13th century. His two major contributions to the study of knowledge that are relevant to AI were nominalism and his insistence on simplicity. With nominalism, he argued that what we know is expressed in language. As a pragmatic principle in AI programming, that translates roughly into the principle that if someone can accurately describe how they solve a problem, then a program can be written to solve it. The principle of parsimony, now known as Occam's Razor, states that *plurality should not be assumed without necessity.* In other words, explanations that mention fewer entities and mechanisms should be preferred to more complex ones.

So, by the time modern science was getting started, several important principles about knowledge had already been clearly established by the ancient Greeks. Medieval philosophers reinforced and added to the early concepts, as shown in table 2.

Pythagoras	Mathematics holds the key to correct descriptions of the world.
Socrates	Seeking knowledge is good. Knowing what we don't know (metaknowledge) is valuable.
Socrates (Plato)	Empirical knowledge is true belief with an account: beliefs have to be justified to be called knowledge.
Aristotle	Observation is a legitimate source of knowledge. Symbolic logic is a means of increasing our store of knowledge through valid inference: Knowledge beyond mathematics can be proved.
Euclid	New knowledge can be derived by rigorous proof.

Table 1. Some Contributions of Early Greek Philosophers to Our Understanding of the Concept of Knowledge.

R. Grosseteste	Mathematics is essential for knowledge of the natural world. Knowledge can be established experimentally.
William of Ockham	Knowledge is linguistic. Simpler expressions of knowledge are preferable.

Table 2. Some Contributions of Medieval Philosophers to our Understanding of the Concept of Knowledge.

Skipping ahead a few more centuries into the 15th and 16th centuries, philosophers continued to investigate scientific questions by elaborating the earlier themes and by making new distinctions. By the 1500s many people were using observation and experimentation (that is, the scientific method) to produce new knowledge about the natural world. In the 17th and 18th centuries scientific discoveries were made at an unprecedented rate (Pledge 1939) and the foundations of modern science were clearly established.

Some of the most relevant principles and ideas to come out of the early modern work are shown in table 3.

Francis Bacon's epigram "knowledge is power" is where we started. Bacon clearly articulated what is now known as the scientific method, the steps of empirical investigation that lead to new knowledge, as opposed to deductive logic "as acting too confusedly and letting nature slip out of its hands." The old logic, he said, "serves rather to fix and give stability to the errors which have their foundation in commonly received notions than to help the search after truth" (Bacon 1620). He is also known for his description of science as a multistep process that may involve teams of people in specialized tasks, such as data gathering. He emphasized planned experiments as an essential step in the inductive process.

René Descartes is most known today for his work on algebra and geometry, but he also wrote about the scientific method (Descartes 1637). In the *Discourse on the Method of Rightly Conducting the Reason and Seeking Truth in the Sciences* he advocates accepting as knowledge only "clear and distinct" truths. He articulated the divide-and-conquer strategy of problem solving and advocated enumerating all possibilities in considering solutions to a problem so as not to overlook any—both powerful ideas for AI.

Sir Isaac Newton's scientific and mathematical contributions, of course, are awe-inspiring. Perhaps less appreciated is his emphasis on clear writing, starting with publication in a contemporary language (English) instead of Latin. As noted by one historian, "Newton established for the first time in any subject more concrete than pure mathematics the strictly unadorned expository style of Euclid, an important advance on the cumbrous and pretentious habits until then current" (Pledge 1939, p. 66).

Newton's method, described in 1669, is one of successive approximations, which he applied to solving polynomials and others

F. Bacon	Knowledge is power. Planned experiments facilitate scientific investigations. Scienctific questions can be investigated by large teams.
R. Descartes	Solve problems by divide-and-conquer. Make complete enumerations to be sure nothing is overlooked.
I. Newton	Clear exposition facilitates knowledge transfer. Iterative refinement is a powerful heuristic for solving hard problems. Crucial experiments can be defined to refute hypotheses, or, if negative, help support them.
T. Bayes	Prior probabilities are important pieces of knowledge. Laws of probability can establish the conditional probability of an event given the evidence.

Table 3. Some Contributions from the Early Modern Period to Our Understanding of the Concept of Knowledge.

have applied to more complex functions. It was also used by Heron of Alexandria (Hero) to find square roots in the first century.[7] His method was one of the first clearly articulated principles of heuristic reasoning, which can be applied to symbolic reasoning as well as mathematics.

Newton added to the concept of scientific reasoning the idea of the "crucial experiment" (Boorstin 1985). This kind of experiment establishes a critical question whose answer, once observed, can falsify a hypothesis. A negative outcome cannot prove the hypothesis, but the absence of refutations is widely accepted as support. Although there are methodological difficulties with the concept, it is still a powerful idea in every science.

Bayes's Theorem is particularly important because it establishes the probability of a hypothesis *H* conditional on evidential support *E* (Joyce 2006). For the first time, scientists had a way to quantify the degree to which evidence supports a hypothesis and a way to revise a degree of belief incrementally as new evidence becomes available. A cornerstone of Bayesian probabilities is that the prior probability of the evidence is an essential starting point.

Some additional principles we can draw on from the early modern period are shown in table 4 without comment.

During the last hundred years or so, philosophers and scientists have continued to add to the list of things we need to pay attention to in implementing the knowledge principle (see table 5). One of the towering giants of the century was Bertrand Russell.

Bertrand Russell and Alfred North Whitehead developed the first-order predicate logic and showed its power for expressing much of what we know. But we also have seen that it needs to be augmented, or replaced, with other tools for representation and reasoning. For example, nonmonotonic reasoning does not obey the laws of first-order logic because new information can negate previously believed propositions. Kurt Gödel's proof that there are some true statements that cannot be proved in first-order logic is not relevant for building knowledge-based systems and may be totally irrelevant to the question of whether AI is possible (contrary to what others have claimed). However, it does underscore the need to consider metalevel knowledge when building an intelligent system.

Russell also showed us that we—or a program—can get into serious logical difficulties when we describe things self-referentially. When Epimenides the Cretan says "All Cretans are liars," we don't know how to assess the truth or falsity of his statement. Also, when we define a class in terms of membership in the same class we lose the ability to assess class membership. For example, Russell's famous barber paradox concerns a barber who shaves all men who do not shave themselves. Logic does not let us answer the question whether the barber himself is in that set or not. Paradoxes such as these are strong reminders that we do not yet know everything about managing knowledge (Quine 1966).

Because of Gottlob Frege's work on intensional predicates, we know that statements about beliefs and knowledge, among other things, do not follow the logical rules allowing substitution of equals for equals. Consider:

> John believes Aristotle invented logic.
> Aristotle was Plato's pupil
>
> . . . but it is not necessarily true that:
>
> John believes Plato's pupil invented logic (because John does not know the relationship between Aristotle and Plato)

Therefore, when we build knowledge-based systems, keeping track of what the program knows and believes is not a simple matter of

G. Leibniz	An object is completely defined by its features (attribute-value pairs) .
I. Kant	Our own conceptual framework determines what we can know about objects in the world.
D. Hume	Knowledge of the natural world is never completely certain. Much of what we know rests on the assumption that the future will be like the past.
J. S. Mill	General laws can be discovered from data. Causality is determined, in part, by systematic variation of an independent variable with observed variation in a dependent variable shortly afterward.

Table 4. Additional Contributions from the Early Modern Period to Our Understanding of the Concept of Knowledge.

deduction. As John McCarthy and others have pointed out, it can be enormously complicated to keep track of a computer's—or a person's—set of beliefs, and revise the set as more information becomes available. Jon Doyle's work on truth-maintenance provides some implementation guidelines to start with.

Carl Hempel clarified the concept of an explanation in a way that is very relevant to a program's offering explanations of its beliefs. Acceptable explanations of a phenomenon *P*—such as a rainbow[8]—under the deductive-nomological model of explanation, follow the form:

Why *P?* (for example, why is there a rainbow?)

If *A* is observed then *P* follows

(If the sun reflects and refracts off water droplets ...)

A is observed

(These conditions exist)

Therefore a rainbow is to be expected

Hempel's model turns out to be intuitively acceptable for the users of knowledge-based systems who want a rationale for a program's advice. It was, in fact, one of the guiding principles behind Mycin's explanation system (Buchanan and Shortliffe 1984).

Thomas Kuhn introduced the concept of a paradigm into discussions of knowledge, extending Immanuel Kant's notion that our beliefs and knowledge structures are framed within a conceptual framework (Kuhn 1962). For Kuhn, a paradigm in science includes not just the vocabulary and conceptual framework of a domain, but the shared assumptions, accepted methods, and appropriate instrumentation of the social community working within that domain. In this sense, the paradigm defines both the legitimacy of questions and the acceptability of proposed answers.[9] When implementing a knowledge-based system, we now understand the importance of making both the ontology and the assumptions as explicit as we can.

Nelson Goodman has examined the kinds of statements that make good general laws—the sorts of statements that describe causal relations and can be used in Hempel's model of explanation. Pure descriptions of things in the world that "just happen" to share a common property may fit into a universal generalization, but they are not good candidates for scientific laws. In Aristotle's terms, these are accidental predicates. For example, the assertion, "All people in this room at this hour today are interested in AI," may be a true description (more or less) of everyone here, but it only "happens" to be true. We could not use it, for example, to predict that a janitor in the room waiting for me to finish is interested in AI. Goodman and others have found difficulties in trying to define criteria for the kinds of predicates that are appropriate for scientific laws. But one criterion we can use operationally in knowledge-based systems and machine learning systems is that the predicates used appropriately in general laws are predicates whose semantics have allowed such use in the past. So a predicate like "reflects light" can be tagged as potentially useful for a theory, while an accidental predicate like "is in this room at this hour" would not be tagged.

There are (at least) two other concepts from contemporary writers outside of AI that are useful for us to remember in building knowledge-based systems. George Polya (1945) introduced the concept of heuristics into our paradigm of reasoning, referring to rules that are not always true but can be very powerful when they are. He was not referring to probabilistic arguments leading to a conclusion that is prob-

G. Frege	Belief statements are intensional.
B. Russell	Predicate logic works. Self-referential statements are problematic.
C. G. Hempel	Explanations appeal to general laws.

Table 5. Some Contributions from the Last Hundred Years to Our Understanding of the Concept of Knowledge.

ably true or about measuring a heuristic's degree of applicability, but to the importance of the rules themselves in making progress toward solving hard problems. One of the sources of ideas for Polya was a diagram of the situation. In a geometry problem, for example, if two line segments look equal in the diagram, it might be useful to try to prove that they are in the general case. For us, the main lessons in Polya's work are that reasoning programs need heuristics and heuristics can be made explicit.

Michael Polanyi emphasizes that much of the useful knowledge a person has cannot be made explicit—in his terms, it is tacit knowledge. In building expert systems by acquiring knowledge from experts, we often run into areas of expertise that are not easily articulated. We might say the expert's knowledge has been compiled. It is good to be cautioned that such knowledge is not easy to acquire. Asking an expert, "How do you solve this kind of problem?" may lead nowhere. But we have also found another tack to be useful in eliciting knowledge: when an expert seems to be struggling to articulate what he or she does, turn the question around and ask, "For this kind of problem, what would you tell an assistant to do?"

Psychologists and economists have also given us some considerations relevant to the implementation of knowledge-based systems. Kahneman, Slovic, and Tversky (1982) collected numerous data showing that human reasoning under uncertainty is not strictly Bayesian. For example, reasoning about a new case often involves comparisons with the most recent cases seen. And people tend to ignore reversion to the mean when assessing their own performance on a task; instead they (we) believe that our best performance has become the new norm and we're disappointed with below-peak output. One lesson to draw from this work is that AI programs need not outperform the best experts to be beneficial: they may help prevent mistakes on the parts of practitioners thus raising the lowest common denominator among people performing a task.

The well-known Pareto Principle states that most effects come from relatively few causes—in quantitative terms, it is known as the 80/20 rule: 80 percent of the benefit from performing a task comes from about 20 percent of the effort. It is an important design consideration for applications programs. A designer can demonstrate an early prototype with high credibility by carefully choosing which 20 percent of the relevant knowledge to put into a system.

Herb Simon's Nobel Prize–winning work in economics on bounded rationality established the fundamental axiom of AI: finding a satisfactory solution is often a better use of resources than searching longer for the optimal solution. This, of course, is the principle of satisficing. Don't leave home without it!

Simon's work with William Chase (1973) demonstrated that expert problem solvers store and use their knowledge in chunks. For example, expert chess players use high-level pattern descriptions, such as king-side attack, instead of descriptions of the individual pieces on the board.

Some recent contributions to our understanding of the concept of knowledge are shown in table 6.

AI WORK

With the invention of digital computers, Alan Turing began the empirical study of how much and what kinds of intellectual work computers can do. Insights from the early work mark the beginning of AI and what is now known as computational philosophy (Thagard 1988). That is, writing programs has become a new way to study the nature of knowledge.

There are several instances of programs using specialized knowledge to achieve high performance before 1970. Arthur Samuel's checker player used knowledge from experts. Geoffrey Clarkson's simulation of an investment trust officer was essentially a collection of rules from one expert. Richard Greenblatt's mid-1960s chess-playing program, MacHack, was based on clearly articulated principles of good chess play, such as the value of controlling the center. Both Jim Slagle's and Joel Moses's programs

T. Kuhn	Scientists work within paradigms (ontology plus assumptions plus …).
N. Goodman	Accidental predicates do not make good general laws.
G. Polya	Heuristics are a powerful form of knowledge.
M. Polanyi	Much of what we know is tacit.
D. Kahneman, P. Slovic, and A. Tversky	Humans are not Bayesians.
V. Pareto	80 percent of the benefit often results from 20 percent of the effort.
H. Simon	Satisficing is often better than optimizing.
H. Simon and W. Chase	Knowledge is chunked.

Table 6. Some Important Recent Contributions to Our Understanding of the Concept of Knowledge.

for solving integration problems symbolically and Tony Hearn's program for algebraic simplification used considerable knowledge of the tricks mathematicians use.

But Edward Feigenbaum's 1968 address to the International Federation for Information Processing (IFIP) Congress was the first clear statement of the knowledge principle in the computer science literature (Feigenbaum 1968), describing the role of knowledge in problem solving, Dendral's problem in chemistry in particular. We believe that Dendral, and then Mycin were the first to fully embrace the knowledge principle. We were deliberately seeking knowledge from experts because we were not ourselves very knowledgeable about the domains. We tried to be systematic about the methods we used for eliciting knowledge from experts (Scott, Clayton, and Gibson 1991). We tried to be disciplined about representing knowledge in simple, uniform data structures that could be understood and edited from the outside or used by the program to explain their reasoning. And we separated the parts of the program that held the knowledge from the parts that reasoned with it.

Guidelines for implementing knowledge-based systems have been suggested before (Davis 1989, Buchanan and Smith 1988), and those are still relevant to building systems. Here I wish to focus specifically on some of the things we have learned about knowledge through observing and experimenting with knowledge-based systems. Since most of these items are familiar to most of you, I present in table 7 the collection as reminders without much explanation.

Existence of Knowledge

First, of course, it is essential that we choose problems for which there is knowledge. If no person knows how to solve a problem, then it is dangerous to promise a computer program that solves it. Predicting terrorist attacks with any precision is a good research problem, for example, but it is not a problem to solve with a deadline.

On the other hand, if it is easy to teach someone how to solve a problem, there is little point in writing a program to solve it. Finding powdered Tang in a grocery story might be facilitated with a table of synonyms and aisle labels, but it hardly requires much inference unless you are in the wrong store.

Search Space

Newton had argued that it was impossible to enumerate hypotheses for interesting problems in science (called "induction by enumeration") because there can be infinite numbers of them. Turing, however, showed that defining a search space—even an infinite one—can provide a conceptual starting point for implementing heuristic search. Defining a start state and a successor function allows a program, in principle, to generate hypotheses and test them one by one until a solution is found or time runs out. Heuristics allow a program to focus on likely parts of the space.

Simon's principle of satisficing works well with heuristic search because it generally broadens the definition of the goal beyond the single-best state. Doug Lenat's AM program broadened the concept of the search space by allowing the generator to be a plausible move

Knowledge of the task domain must exist.
Defining a search space is essential.
Explicit knowledge structures facilitate system building.
Common sense can be defined or assumed.

Table 7. Some Conclusions about Knowledge from Early Work in AI.

generator, as opposed to an in-principle complete generator as in Dendral.

Explicit Representation

John McCarthy's early paper "Programs with Common Sense" (McCarthy 1958) argues for representing knowledge explicitly in declarative statements so that it can be changed easily from the outside. Although the distinction between declarative and procedural knowledge is not a sharp one, users of programs are quick to tell us when editing a knowledge base is easy or hard.

Common Sense

Programs that solve real-world problems must deal with numerous assumptions about the way the world works. McCarthy has argued that a program needs access to the deductive consequences of a large axiom set. Lenat and Feigenbaum have argued that common sense can be encoded in millions of individual facts and relations that keep a program from looking silly. After McCarthy criticized Mycin for failing to know that amniocentesis would not be performed on a male patient or that dead people don't need antibiotics, we came to realize that we had an effective operational way of dealing with common sense other than trying to encode it. Namely, we assumed that users of a knowledge-based system would themselves have some common sense. Doctors and nurses using Mycin, for example, could be presumed to know when their patients are already dead.

One important class of knowledge-based system shares the load of problem solving so that the system assists a person as a partner but depends on the human user to provide additional knowledge of the problem area. In addition to users providing some of the common sense, users are likely to have important input when it comes to values and ethical considerations in choices. That is, depending on the context of use, a program may be able to depend on a human partner to provide commonsense knowledge.

Additional lessons about knowledge relevant to AI are presented in table 8.

Solved Cases, Generalizations, and Prototypes

Arthur Samuel's checker-playing program clearly demonstrated the power of rote learning. If a problem has been successfully solved, look up the solution and keep adding to the library of solved cases to become smarter with experience.

Case-based reasoning (CBR) is another way to use previously solved cases. Instead of finding an exact match for a case in a library and transferring a solution directly, as in rote learning, CBR systems find similar matches and modify their solutions.

Marvin Minsky and Roger Schank have advocated encoding frames of prototypical objects and scripts of prototypical events. Knowledge about prototypical members of a class helps us fill in what we need to know, understand what we don't know, and make good guesses about missing information. Knowledge about how complex situations typically unfold in time helps us in just the same ways. Both frames and scripts can give a program expectations for what to expect and default values for filling in missing information.

Items of Knowledge

Three decades of experience with knowledge engineering has taught us that knowledge does not come in prepackaged "nuggets." Eliciting knowledge from experts is not so much like mining for gold as like coauthoring a textbook. It is a social process whose success depends on the personalities and interpersonal skills of both the knowledge engineer and expert.

We also know that the chunks of knowledge in a knowledge base are not independent, and therefore large sets of knowledge items can interact in ways that are not completely predictable. For this reason, it is useful to collect items into nearly independent groups, where the interactions within a group are easier to see and the interactions across groups are infrequent (table 9).

Rules

Emil Post proved that simple conditional rules together with modus ponens constitute a logically complete representation. However, one lesson from work with rule-based systems is that rules are more powerful when augmented with additional knowledge structures, such as definitions and class-subclass hierarchies.

Whatever the data structures that are used to represent knowledge, however, an important lesson learned with some pain is to use the concepts and vocabulary of the end users. Experts in the home office or academe do not

necessarily speak the same language as users in the field. It is also critical that users provide knowledge about the work environment. There is a danger that the expert providing the knowledge is unfamiliar with the constraints and assumptions implicit in the context of the actual work place.

Hierarchies

Aristotle's insights on organizing knowledge hierarchically have been implemented in semantic networks, frames, prototypes, and object-oriented systems in general. Knowing the class to which an object belongs gives us considerable information about the object. Thus we save time and space by knowing about classes. Moreover, we gain the ability to make plausible inferences about an object in the absence of details, since it is usually safe to assume information inherited from a class is correct unless we're told otherwise.

Default Knowledge

Giving a program a set of defaults, or a way to determine them, broadens the range of problems the program can deal with. Even in the absence of specific values for some attributes, a program can find plausible solutions if it can make reasonable guesses about the missing values. Default values may be stored in prototypes, hierarchies, definitions, or lists.

Uncertainty

The knowledge we use for problem solving is frequently uncertain and incomplete. Also, the information we have about the objects and events named in any individual problem is uncertain and incomplete. Yet we, and the systems we build, are expected to provide satisfactory solutions with the best information available. Fabricating information to fit a preconceived conclusion is not scientifically acceptable. Therefore, implementing the knowledge principle requires implementing a means of dealing with uncertain and incomplete knowledge.

The simplest way to deal with this inconvenience is to ignore it: act as if everything known to the program is categorically certain. For simple problems this is often sufficient. Currently, the most popular way of dealing with uncertainty is to assign probabilities to what is known and combine them with Bayes's Theorem. Nonprobabilistic methods, like contingency tables, fuzzy logic, and Mycin's certainty factors, have also been used successfully.

Attached Metadata

A large knowledge base needs to be constructed incrementally. Often more than one expert and more than one knowledge engineer are involved in the construction and modification. For these reasons, it is good practice to tag the identifiable chunks of knowledge with additional information. In Mycin, for example, it was useful to note the author of the chunk, the date it was added, subsequent modification dates, literature references, and reasons why the chunk was added.

More lessons about knowledge relevant to AI are presented in table 10.

5	Knowledge of past cases is powerful.
6	Generalizations and prototypes are powerful.
7	Nearly-independent chunks can be defined.
8	Elicitation of knowledge is social.

Table 8. Additional Lessons about Knowledge Relevant to AI.

9	Rules, hierarchies, and definitions complement one another.
10	Default knowledge adds breadth and power.
11	Uncertainty and incompleteness are ubiquitous.
12	Metadata facilitates debugging and editing.

Table 9. Additional Lessons about Knowledge Relevant to AI.

Constraints

Many problems require synthesizing a solution that satisfies a large number of constraints. Scheduling problems, configuration problems, and design problems all have this character, each with powerful models for how to solve them. Each constraint is a specific piece of knowledge about the problem that can be examined and reasoned about. For example, a program can decide to postpone trying to satisfy a constraint until more information becomes available if the constraint is an explicit chunk of knowledge and not a complicated set of program statements.

Temporal Relations

James Allen's interval calculus, based on Russell's interval definition of time, is a workable set of inference rules for reasoning about the relative times of events. It should be in every knowledge engineer's toolbox for problems in which temporal relationships are important.

13	Constraint-based reasoning is powerful.
14	Temporal reasoning may be required.
15	Analogies may help.
16	Diagrams aid in elicitation — and in future reasoning.

Table 10. More Lessons about Knowledge Relevant to AI.

17	Knowledge can be used opportunistically.
18	Redundancy is often helpful.
19	Knowledge sharing adds power.

Table 11. Still more Lessons about Knowledge Relevant to AI.

Analogies

Analogical reasoning can be a powerful tool, but we still do not understand how to tell a good analogy from a bad one. For example, is the Iraq war like the Vietnam war or not? There is no question that knowledge of similar situations can be helpful, but for the moment case-based reasoning seems to be more tightly constrained than broader analogical reasoning.

Diagrams

Polya advocated drawing a diagram to help us understand a problem and see relevant interrelationships among objects. Unquestionably, a diagram contains information that is useful—as we say, a picture is worth a thousand words. Gelernter's geometry program did use diagrams, and a few others have, but tools for using diagrams are still missing from the knowledge engineer's toolbox.

Table 11 lists still more lessons about knowledge that are relevant to AI.

Opportunistic Use

Oliver Selfridge's Pandemonium program and subsequent work with the blackboard model show that knowledge need not be used in a fixed sequence. It makes good sense to use the items of knowledge that have the most to contribute to a problem at each moment. In scheduling and planning problems, for instance, some constraints are more useful as more of their preconditions are met—which means either waiting until more information is available or forcing the program to chase after the preconditions.

Redundancy

Contrary to Occam's Razor, there are good reasons to encode the same information redundantly in a program. For one thing, we cannot achieve total independence of evidence statements in real-world systems because the components are interrelated. Many things can cause a fever, for example, and a fever can be evidence for many conditions. Also, the actual description of a problem is generally missing some items of data. If there are multiple paths for making inferences, the missing data will not hurt as much.

Shared Knowledge

Reid Smith has suggested that the simple statement of the knowledge principle be expanded to:

$$\text{Power} = \text{Knowledge}^{\text{Shared}}$$

where the exponent indicates the number of people or machines whose knowledge is being brought to the problem.[10] This is an excellent point. It fits with the common wisdom that two heads are better than one. A community of more or less independent problem solvers is a powerful model, if the environment fosters and rewards sharing. Selfridge made an early demonstration of the power of this model in his Pandemonium program, and it has been generalized in the blackboard model, but only recently has it become common in the business world. For example, an article in *The Economist* (May 8, 2003) describes a survey of knowledge management that "found that the best-performing companies were far more likely than the worst-performing ones to use creative techniques for acquiring, processing and distributing knowledge."

The benefits of collaboration among machines have been demonstrated in a few research projects, but collaborative problem solving through knowledge sharing is still far from commonplace in AI. We have a lot to learn about implementing collaborative programs. Parallel searching in search engines like Google enables rapid response, but it is more a result of divide-and-conquer than it is of better thinking through collaboration.

What Does a Programmer Have To Do?

Every problem-solving or decision-making program requires knowledge of the problem area. When we write a program to calculate an area as length times width, we are codifying a piece of knowledge from geometry in the procedure. However, the discipline of knowledge-based

1	Explicit representation of knowledge — within the users' conceptual framework.
2	Explicit reasoning steps.
3	Modular, nearly-independent, chunks of knowledge.

Table 12. Operationalizing Flexibility.

A Primary Lesson from the Mycin Experiments (Buchanan and Shortliffe 1984).

programming goes beyond "hard wiring" relevant knowledge in procedures. Actually implementing the knowledge principle in a program requires codifying the knowledge in ways that it can be examined and changed easily. Ideally, the program can even explain what knowledge it uses to solve a particular problem and why it uses it.

In our description of the Mycin experiments, Shortliffe and I attempted to find a one-word summary of what made Mycin work as well as it did (Buchanan and Shortliffe 1984). The one word was *flexibility*. In the design and implementation of the program, we looked for ways of incorporating knowledge about the domain of medicine and the task of diagnosis in the most flexible ways we could. In particular, we wanted the knowledge to be represented to allow straightforward credit assignment, easy examination and modification, and user-friendly explanations of why the program believed what it did. One simple reason why this is desirable is that complex knowledge-based systems need to be constructed through a process of iterative refinement (recall Newton's method).

Staying flexible seems to have three components at the implementation level that I believe facilitate constructing and fielding successful knowledge-based systems (table 12): (1) explicit representation of knowledge within the program using the vocabulary and procedures of practitioners in the task domain—and relating them under a coherent model of the task domain; (2) explicit reasoning steps that the program can play back ("explain") in an understandable fashion; and (3) modular, nearly independent chunks of knowledge, whether that knowledge is codified in procedures or data structures.

Conclusions

On the assumption that an audience will remember only seven plus-or-minus two things, I have three take-home lessons.

First, Bacon is right: knowledge is power. Philosophers have given us many distinctions in what we know about knowledge. For example, what kinds of evidence count as justifications for a belief, what kinds of knowledge there are, and how distinguishing metalevel and object-level knowledge clarifies a problem.

Perhaps the most important lesson is that real-world problems require real-world knowledge. Problems are hard when relevant knowledge is not neatly codified. Knowledge of objects and processes in the real world is often approximate, ambiguous, incomplete, and wrong. That is one of the most important reasons why human problem solvers require considerable training and experience to solve real-world problems and why assistance from computers is welcome. It is also one of the most important reasons why precise, logical formulations of real-world problems are often inaccurate. And it is the reason why satisficing is a good strategy.

Second, Feigenbaum is right: *more* knowledge is more power. The more kinds of things a program knows about in a domain, the greater the scope of its capabilities. By using class hierarchies, there is considerable economy in adding knowledge about new instances.

Including more knowledge of the objects and events in a problem domain and more knowledge of special cases will increase the power of a program. Computer scientists have shown how to implement many specialized kinds of knowledge needed to solve real-world problems. These include spatial and temporal relationships, conceptual hierarchies, generalizations with and without uncertainty, exceptions, and analogies.

Third, discipline is required to maintain flexibility—implementing the knowledge principle requires making knowledge explicit and modular. Large knowledge bases are constructed iteratively, with an experimental cycle of testing and modifying. Without the flexibility to examine and modify a program's knowledge, we are unable to determine what to change.

Worse yet, without the flexibility to examine and modify a program's knowledge, we are at the mercy of the program's power to take the program's advice without understanding how it is justified. Plato said that's a bad idea.

Let me conclude with a Native American proverb that I hope program designers and future AI programs will be able to operationalize:

> If we wonder often, the gift of knowledge will come.

Acknowledgements

I am indebted to many colleagues who have, over the years, given me the benefit of their own knowledge of philosophy and AI. Please credit them with trying. In particular, Ed Feigenbaum and Reid Smith provided substantial comments on an earlier draft and, among other things, encouraged me to simplify. Phil Cohen provided several comments and the reminder about Russell's interval definition of time.

Notes

1. This article is based on the Robert S. Engelmore Award lecture, which I was honored to present at the Innovative Applications of Artificial Intelligence conference held in July 2006 in Boston, Massachusetts.

2. As many of you know, Bob Engelmore was dedicated to the clear exposition of what we know and equally dedicated to keeping silent when we don't have the facts. Many times when I began hand-waving, Bob would just give me an incredulous look. So it was with some misgivings that I chose the somewhat immodest title, "What Do We Know about Knowledge?"

3. This is admittedly a whirlwind tour, with more reminders than explanations. I hope it not only will remind us of some of the intellectual roots of AI but may also help us create more robust systems in the future.

4. This was anticipated by the Greek poet Menander, who wrote, "In many ways the saying 'Know thyself' is not well said. It were more practical to say 'Know other people'" (Thrasyleon Fragment).

5. There are, of course, notable exceptions of good experimental science and clear documentation in AI. See, for example, Leake (2006). The case for AI as an experimental science is made in Buchanan (1994).

6. Western science owes much to the Arab world, as acknowledged, for example, by White (1967): "The distinctive Western tradition of science, in fact, began in the late 11th century with a massive movement to translate Arabic and Greek scientific works into Latin."

7. See Answers: Newton's Method, www.answers.com/topic/newton-s-method (2006).

8. See the National Center for Atmospheric Research (http://eo.ucar.edu/rainbows/).

9. Bob Engelmore and I were once buying food in a health food store for a backpacking trip. We assumed that the powdered drink Tang would be shelved with other fruit drinks and asked an earnest young clerk where to find it. It was not a legitimate question in that store and we ended up with a lecture on food additives. Thereafter, the word *Tang* was shorthand for discovering a context in which a lecture is a proper response to a question, rather than a simple answer.

10. See R. G. Smith, www.rgsmithassociates.com/ Power.htm.

References

Bacon, F. 1597. Religious Meditations, Of Heresies. In *Essayes. Religious Meditations. Places of Perswasion and Disswasion. Seene and Allowed (The Colours of Good and Evil).* London: John Windet.

Bacon, F. 1620. *The New Organon.* London: Cambridge University Press, 2000.

Boorstin, D. 1985. *The Discoverers*. New York: Vintage Books.

Buchanan, B. G. 1994. The Role of Experimentation in Artificial Intelligence. *Philosophical Transactions of the Royal Society* 349(1689): 153–166.

Buchanan, B. G., and Shortliffe, E. H. 1984. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* New York: Addison Wesley.

Buchanan, B. G., and Smith, R. G. 1988. Fundamentals of Expert Systems. In *Annual Review of Computer Science 3,* 23–58. Palo Alto, CA: Annual Reviews, Inc.

Cornford, F. 1935. *Plato's Theory of Knowledge: The Theaetetus and the Sophist of Plato.* London: Routledge and Kegan Paul.

Davis, R. 1989. Expert Systems: How Far Can They Go? Part One. *AI Magazine* 10(1): 61–67

Descartes, R. 1637. *Discourse on the Method of Rightly Conducting the Reason and Seeking Truth in the Sciences*. Translated by John Veitch. Edinburgh: W. Blackwood and Sons, 1870.

Feigenbaum, E. A. 1968. Artificial Intelligence: Themes in the Second Decade. *IFIP Congress* (2), 1008–1024. Laxenburg, Austria: International Federation for Information Processing Congress.

Joyce, J. M. 2006. Bayes' Theorem. Stanford Encyclopedia of Philosophy. Stanford, CA: CSLI, Stanford University. plato.stanford.edu/entries/bayes-theorem/#3 (June 2006).

Kahneman, D.; Slovic, P.; and Tversky, A., eds. 1982. *Judgment under Uncertainty: Heuristics and Biases*. Cambridge: Cambridge University Press.

Kuhn, T. 1962. *The Structure of Scientific Revolutions*. Chicago: Univ. Chicago Press.

Leake, D.; and Sorriamurthi, R. 2004. Case Dispatching Versus Case-Based Merging: When MCBR Matters. *International Journal of Artificial Intelligence Tools* 13(1): 237–254.

McCarthy, J. 1958. Programs with Common Sense. In *Proceedings of the Teddington Conference on the Mechanization of Thought Processes,* 77–84. London: Her Majesty's Stationary Office.

Pledge, H. T. 1939. *Science Since 1500*. London: His Majesty's Stationery Office.

Polya, G. 1945. *How To Solve It*. Princeton, NJ: Princeton University Press.

Quine, W. V. O. 1966. *The Ways of Paradox and Other Essays*. Cambridge: Harvard University Press.

Scott, A. C.; Clayton, J.; and Gibson, E. 1991. *A Practical Guide to Knowledge Acquisition.* Boston: Addison Wesley.

Simon, H. A., and Chase, W. G. 1973. Skill in Chess. *American Scientist* 61(4): 394–403.

Steup, M. 2006. Analysis of Knowledge. Stanford Encyclopedia of Philosophy. Stanford, CA: CSLI, Stanford University. plato.stanford.edu/entries/knowledge-analysis/#JTB.

Thagard, P. 1988. *Computational Philosophy of Science*. Cambridge, MA: The MIT Press.

White, Jr., L. 1967. The Historical Roots of Our Ecologic Crisis. *Science* 155(3767), (10 March, 1967): 1203.

Bruce G. Buchanan was a founding member of AAAI, secretary-treasurer from 1986–1992, and president from 1999–2001. He received a B.A. in mathematics from Ohio Wesleyan University (1961) and M.S. and Ph.D. degrees in philosophy from Michigan State University (1966). He is university professor emeritus at the University of Pittsburgh, where he held joint appointments with the Departments of Computer Science, Philosophy, and Medicine and the Intelligent Systems Program. He is a fellow of the American Association for Artificial Intelligence (AAAI), a fellow of the American College of Medical Informatics, and a member of the National Academy of Science Institute of Medicine. His e-mail address is buchanan@cs.pitt.edu.