by whether they are understandable and easily changed. Thus computer tools that bring computational leverage to programming are helping computer scientists to regain a sense of control over systems that have become increasingly complex.

REFERENCES AND NOTES

1. J. McCarthy, *Commun. ACM* **3**, 185 (1960).
2. E. Charniak, C. Riesbeck, D. McDermott, *Artificial Intelligence Programming* (Erlbaum, Hillsdale, NJ, 1980).
3. R. Davis, *Science* **231**, 957 (1986).
4. N. Wirth, *Programming in Modula-2* (Springer-Verlag, New York, 1985).
5. B. Wichman, *Commun. ACM* **27**, 98 (1984).
6. G. Birtwistle, O. Dahl, B. Myhrhaug, K. Nygaard, *Simula Begin* (Auerbach, Philadelphia, 1973).
7. A. Goldberg and D. Robson, *Smalltalk-80, The Language and Its Implementation* (Addison-Wesley, Reading, MA, 1983).
8. M. Stefik and D. G. Bobrow, *AI Magazine* **6**, 40 (1985).
9. D. G. Bobrow and M. J. Stefik, *The Loops Manual* (Xerox Corporation, Palo Alto, CA, 1983).
10. M. Stefik, D. G. Bobrow, K. Kahn, *IEEE Software* **3**, 10 (1986).
11. J. Doyle, *Artif. Intell.* **12**, 231 (1979).
12. R. A. Kowalski, *Commun. ACM* **22**, 424 (1979).
13. T. Moto-oka, Ed., *Fifth Generation Computer Systems* (Elsevier/North-Holland, Amsterdam, 1982).
14. R. Davis and J. King, in *Machine Intelligence*, E. Elcock and D. Michie, Eds. (Wiley, New York, 1976), vol. 8, pp. 300–332.
15. D. Waterman and F. Hayes-Roth, Eds., *Pattern-Directed Inference Systems* (Academic Press, New York, 1978).
16. B. G. Buchanan and E. H. Shortliffe, *Rule-Based Expert Programs: The MYCIN Experiments of the Stanford Heuristic Programming Project* (Addison-Wesley, Reading, MA, 1984).
17. J. Gordon and E. H. Shortliffe, *Artificial Intelligence* **26**, 323 (1985).
18. S. Mittal, C. L. Dym, M. Morjaria, in *Applications of Knowledge-Based Systems to Engineering Analysis and Design*, C. L. Dym, Ed. (American Society of Mechanical Engineers, New York, 1985), p. 99.
19. B. Hayes-Roth, *Artif. Intell.* **26**, 251 (1985).
20. I. E. Sutherland, thesis, Massachusetts Institute of Technology, Cambridge (1963).
21. A. Borning, *ACM TOPLAS* **3**, 353 (1981).
22. G. Steele, thesis, Massachusetts Institute of Technology, Cambridge (1980).
23. H. Abelson and G. Sussman, *Structure and Interpretation of Computer Programs* (Massachusetts Institute of Technology Press, Cambridge, 1985).
24. R. Fikes and T. Kehler, *Commun. ACM* **28**, 904 (1985).
25. D. G. Bobrow, *IEEE Trans. Software Eng.* **SE:11**, 10 (1985).
26. D. Bobrow, K. Kahn, G. Kiczales, L. Masinter, M. Stefik, *CommonLoops, A Graceful Merger of Common Lisp and Object-Oriented Programming* (Xerox Corporation, Palo Alto, CA, 1985).
27. B. Hailpern, *IEEE Software* **3**, 6 (1986).
28. A. Goldberg, in *Interactive Programming Environments*, D. Barstow, H. Shrobe, E. Sandewall, Eds. (McGraw-Hill, New York, 1984), p. 141.
29. B. Sheil, *ibid.*, p. 19.
30. W. Teitelman and L. Masinter, *ibid.*, p. 83.
31. D. Barstow, *AAAI Magazine* **5**, 5 (1984).
32. C. Rich and H. Shrobe, in *Interactive Programming Environments*, D. Barstow, H. Shrobe, E. Sandewall, Eds. (McGraw-Hill, New York, 1984), p. 443.
33. J. S. Brown, R. Burton, J. de Kleer, in *Intelligent Tutoring Systems*, D. Sleeman and J. S. Brown, Eds. (Academic Press, New York, 1983), p. 227.
34. T. Winograd, *Commun. ACM* **22**, 391 (1979).
35. M. Sanella, *Interlisp-D Reference Manual* (Xerox Corporation, Palo Alto, CA, 1983).

36. We thank J. S. Brown, J. de Kleer, K. Kahn, G. Kiczales, M. Miller, and J. Shrager for comments on earlier versions of this paper.

Knowledge-Based Systems

RANDALL DAVIS

First developed two decades ago, knowledge-based systems have seen widespread application in recent years. While performance has been a strong focus of attention, building such systems has also expanded our conception of a computer program from a black box providing an answer to something capable of explaining its answers, acquiring new knowledge, and transferring knowledge to students. These abilities derive from distinguishing clearly what the program knows from how that knowledge will be used, making it possible to use the same knowledge in different ways.

WORK IN ARTIFICIAL INTELLIGENCE (AI) HAS OFTEN looked for inspiration to the only easily accessible example of intelligence, human behavior. The earliest attempts to design intelligent programs were heavily influenced by the observation that people seem to make some progress on virtually any task, even those that are unfamiliar. Given problems in symbolic logic or algebra, naïve subjects displayed a consistent set of widely applicable problem-solving methods (*1*). Generality came to be seen as a keystone of human intelligence; intelligence appeared to reside in a small collection of domain-independent problem-solving methods. Programs based on such methods displayed encouraging early success.

It became clear that although these methods provided a useful foundation, they were soon overwhelmed by the complexity of real-world problems. Performance on such problems seemed to require large stores of task-specific knowledge (*2*).

This observation led to a significant shift in emphasis for the part of the field that came to be known as knowledge-based systems, in which work has come to focus on the accumulation, representation, and use of knowledge specific to a particular task. The term knowledge-based is primarily a label for this focus and an indication of the source of the systems' power: task-specific knowledge, rather than the domain-independent methods used in early AI programs (*3*). That knowledge is often incomplete and at times involves inexact judgments, unlike the knowledge that underlies carefully designed algorithms of traditional software. The systems can also be characterized by an architecture and a set of capabilities that result from it, including explanation, knowledge acquisition, and tutoring, as well as problem-solving performance.

Previous discussions have focused largely on performance, describing applications and levels of performance reached (*4*). This discussion considers how these systems have expanded our view of a program, an expansion made possible in part by the ability to use the same knowledge in several different ways. This is demonstrated in the context of a rule-based system, since it is the most familiar technology used for constructing these systems.

Randall Davis is an associate professor at the Sloan School of Management at Massachusetts Institute of Technology and a member of the MIT Artificial Intelligence Laboratory, Cambridge 02139.

The Underlying Technology

Two components are central to the operation of these systems: the knowledge base and the inference engine (*5*). The knowledge base contains all of the system's task-specific information, often in the form of a few hundred simple rules of the form shown below. The inference engine is the system's machinery for selecting and applying knowledge from the knowledge base to the specific case at hand.

The knowledge base. A variety of AI technologies have been used in constructing knowledge bases, including rules (*6*), frames (*7*), semantic nets (*8*), predicate calculus (*9*), and procedures (*10*). A rule-based system is used here for illustration, since it is one of the most common and easily understood approaches. A sample rule from the Mycin system (*6*) (designed for diagnosing infectious diseases) is shown below. Mycin's knowledge base is composed of some 500 rules like this.

Rule 27

If the infection is primarily in the blood stream, and
the culture was obtained from a normally sterile site, and
the suspected portal of entry of the organism is the gastrointestinal tract,

then there is suggestive evidence (.8) that the identity of the organism is bacteroides.

The rules are for the most part quite simple: the if-then form provides modest machinery for expressing knowledge. Despite this, the successful application of this technology often results in rules that are relatively self-contained and coherent. Rule 27, for example, can be understood on its own, without examining all the other rules in the system, indicating that it proved possible to "dissect" knowledge of infectious disease diagnosis into a few hundred relatively independent rules. The rules are also comprehensible because they have been supplied by human experts asked to explain their own reasoning; hence they use a familiar vocabulary and reasoning style.

The .8 indicates how strongly the conclusion follows from the premises. Since there are few absolute rules, there must be some way to express "maybe" and "probably," and some way to "add" them together. This is a topic of considerable research (*11*) and little agreement. It may, however, be of only secondary importance; most of a system's performance seems to arise from having the rules at all, that is, knowing which facts lead to which conclusions. In one experiment (*12*) "rounding off" the certainty measures in a set of rules from nine different fine-grained values to only two (effectively "certain" or "possible") actually resulted in a small increase in system accuracy.

The inference engine. The most common forms of inference engine use the rules to reason either forward, from observations to conclusions, or backward, from a hypothesis back toward observations that might support or refute the hypothesis.

Systems that reason forward start with data supplied by the user and apply all rules whose "if" part are satisfied. If, for example, the input data given to the system included the facts that the infection was in the blood stream, the culture was from a sterile site, and the portal of entry was the gastrointestinal tract, the system would note that rule 27 was applicable and would draw the conclusion shown. This may in turn result in a cascade of rule applications, as one rule produces a conclusion needed in the premise of another.

Systems that reason backward start with a goal (such as, determine the identity of the infecting organism) and retrieve all rules whose conclusions deal with this topic (*13*). Each premise of the rules retrieved becomes a new goal in turn, and rules relevant to those goals are retrieved. In this case, reasoning backward through rule 27 would cause the retrieval of rules able to infer infection type, site sterility, and portal of entry. This process of working backward from initial goal toward more primitive data continues until the system encounters topics for which there are no rules. At that point information is requested from the user.

It is also possible to combine both procedures, reasoning forward through the rules from an initial set of observations to generate tentative hypotheses, then working backward, through other rules, from the hypotheses to request additional observations or tests from the user.

Although rule-based systems have been built for a wide variety of tasks, the implication to be drawn is not that rules are a powerful way to represent knowledge, nor that applying rules is a powerful reasoning method. The technology plays an important supporting role by encouraging the style of knowledge encoding illustrated by rule 27, a style that has facilitated the difficult process of making explicit the knowledge that had previously existed implicitly in the practice of experts. More important, the existence of these systems is an interesting comment about some varieties of expertise: it is an empirical observation that substantial bodies of skill have been captured by accumulating a few hundred or a few thousand simple chunks. It was not obvious at the outset that any interesting problem would succumb to such a simple dissection. It is primarily a comment about the knowledge in an area when such an approach succeeds and interesting that it has done so in more than a few isolated cases.

While the performance of these systems has been impressive in some cases, they also have well-recognized limitations. The task needs to be primarily cognitive, for instance; perceptual or motor tasks require research procedures from other areas of AI (*14*). These systems are appropriate where the task is more an art (like designing organic synthesis), than a science (like solving simple physics problems); in the latter case straightforward algorithms suffice. The task should be a narrowly defined area in which human expertise has been demonstrated. The system may perform well within this area, but may fail on tasks that appear to us only superficially different, giving important evidence of the shallowness of understanding of most existing systems. For example, while rule 27 mentions the gastrointestinal tract, to the system this is simply the name of one of the portals by which bacteria can enter the body; it knows nothing of food, digestion, anatomy, and so forth.

"What to Know" as Opposed to "What to Do"

One of the distinguishing characteristics of these systems is the sharp distinction between the inference engine and the knowledge base. This division has two interesting consequences. First, it makes possible the substitution of a new knowledge base for a new task in place of the existing knowledge base, producing a new system as a result. Experimental systems for tasks as diverse as pulmonary function test interpretation (*15*), finite element analysis (*16*), and electronic troubleshooting (*17*) were all constructed by assembling different sets of rules for the same inference engine (*18*).

Second, it encourages taking an additional step along what has been called the "how to what" spectrum. Programming in machine or assembly language requires attending to a great many details (like register allocation) specifying exactly how a computation should be carried out. Higher level languages like Fortran or Algol suppress this detail, offering a new set of constructs for expressing computation at the level of arithmetic expressions, subroutine calls, and iteration. This facilitates expressing what should be done (what expressions to compute) with considerably less attention to exactly how it should be done. Current database retrieval languages take a

similar step, allowing the user to express what to retrieve (such as "Show all the salesmen over quota in New England") rather than writing a program specifying how to find it in the database.

The separation of inference engine and knowledge base produces another step in this direction, encouraging the construction of a knowledge base containing what the program should know, rather than what it should do (*19*). That is, the key task is to express knowledge about the problem (individual rules), without also committing to one specific way of using that knowledge (as, for example, a decision tree). The most important consequence of this distinction is that it enables the system to make multiple different uses of the same knowledge, facilitating explanation, knowledge acquisition, and tutoring. The separation of inference engine and knowledge base, and resulting multiple use of the same knowledge, is thus at the root of an expanded view suggesting that a program may do considerably more than compute an answer. One relatively easy way to make clear how this can work is to examine the spreadsheet programs currently popular in business, using them as a familiar example to illustrate a number of important technical ideas.

Spreadsheets

Spreadsheet programs are a particularly convenient way of doing arithmetic computations. They offer a way of expressing the calculation in terms of a two-dimensional array of "cells," where the value in each cell is either given by the user, or computed from values in other cells according to the user's instructions (Fig. 1). In computing the budget for an academic research project, for example, the first three cells in column 1 might be labeled faculty salary, graduate student salary, and support staff salary, with the fourth cell labeled salary subtotal, and its value indicated as the sum of the values in the first three cells. If we then insert values into the first three cells, the program will compute (and display) the value in the fourth cell. If we change any input value, we can ask for an updated view of the array.

In any real use elaborate computations are done that contain hundreds or even thousands of rows and columns. The important point is that these programs change the nature of the task from writing a program, to building a model, that is, expressing relationships.

This happens because many of the "details" have been taken care of. Most important of these is control: in any traditional programming language we need to specify the sequence of steps to be carried out. In a spreadsheet we can focus instead on expressing the (mathematical) model itself. One important result is a clear distinction between what the model says (relations between cells) and how the model is to be used. Thus, like knowledge-based systems, spreadsheets distinguish between what to know and what to do.

Knowledge expressed in this fashion can be used in several different ways. For example, spreadsheets are traditionally used to reason forward, working from input data toward results. But it is also useful to be able to work "backward" on the same model, perhaps to keep one or more category of expense within known limits. Given a fixed amount of money available for salaries, for example, we may wish to create a budget that matches. One way to do this would be to run the model forward in a process of trial and error: supply a set of inputs, run the model, check for overruns, then try changing some inputs to eliminate the overruns. But in any fairly complex model, it may not be obvious which inputs to change (which inputs affect the quantity in question?) and how (up or down, and by how much?).

It is more effective to work backward: start from the offending amount and by examining the equation that produced it, determine

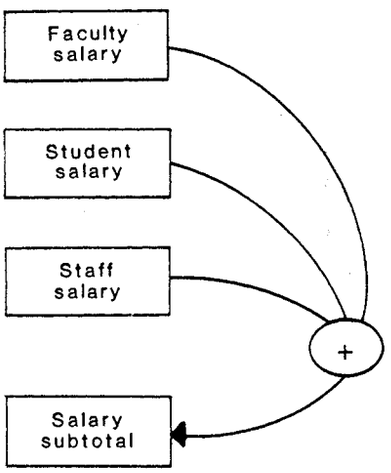

Fig. 1. Schematic view of the operation of a spreadsheet.

what quantities it depends on and how. Next see what each of those quantities in turn depends on. This process continues until we arrive at primary inputs (that is, values input by the users, because there are no equations for them). At that point we will have accumulated a picture of which inputs are relevant and how they should be modified (*20*). This is a new way of using exactly the same model: instead of running it forward, the system works backward through the same information.

Now imagine trying to get this same effect in any traditional program. Having written a Fortran program for the budget, for example, all we have to do is run it backward. This makes no sense, of course, but why? Because a Fortran program is fundamentally a description of a set of calculations to carry out, not a description of relationships between variables. In order to express something in Fortran, the programmer has to write it as a calculation, deciding for instance which should be the independent variables (faculty, student, and staff salary) and which the dependent variables (salary subtotal). As a result, the program doesn't express what we know (the relations in the model), but requires that we state what to do (in this case, run the model forward). By keeping these two things distinct we leave open the option to use the knowledge in multiple ways.

This distinction also facilitates explanation. In a complex model we might find it useful to be able to answer questions about the results, such as

How did you determine that overhead would be $126,344?
Why did the total decrease by $43,250 when I lowered graduate student support by only $18,155?
How is the amount of employee benefits computed?

The first two are answered by augmenting the machinery that runs the model, having it maintain a record, an audit trail, of the computations it carries out. Given any result, it can then review the audit trail, describing each computation. The third question is answered not by running the model but by examing it, retrieving the equations relevant to the computation mentioned. This is a third use of the same information: the knowledge base is in effect being treated as a database containing information to be retrieved (*21*).

An additional advantage of the distinction is clarity: the content of the model is more obvious in the equations of the spreadsheet than in Fortran code. This phenomenon is well known: the body of a technical paper never lists code, it describes the relevant equations. This distinction also offers increased assurance that the program and model express the same thing because they are the same thing.

As this example suggests, there are significant advantages to distinguishing carefully between expressing the knowledge underlying a task and writing a more traditonal program. This is of interest because everything true for the mathematical computations done with a spreadsheet is also true for the symbolic inferences done with knowledge-based systems. In particular, the task changes from writing a program to specifying the knowledge required for solving the problem. Like the spreadsheet, a knowledge-based system is constructed not by describing an algorithm, but by describing things about the world. With the spreadsheet this information was written in terms of equations and mathematical relations; for a knowledge-based system the information consists of symbolic relations, that is, task-specific rules.

This approach of telling a program what to know accounts in part for the difference between knowledge-based systems and traditional software. A second difference is highlighted by considering what basic constructs each supplies and how those constructs affect our view of the system's behavior. As noted, high-level languages like Fortran supply constructs such as subroutine calling and iteration. As a result, while it is certainly possible for the programmer to build those constructs when using assembly language, using a high-level language makes it unnecessary, thereby reducing both the workload and opportunity for error. The new vocabulary also encourages viewing the operation of the system in terms of those higher level constructs: we can think in terms of formulas evaluated or subroutines called, rather than individual additions and multiplications. Knowledge-based systems in turn supply another set of constructs, like forward- and backward-reasoning inference engines and trace facilities to aid explanation. The consequences are similar: the workload and chance of error are reduced, and more importantly, there is a new vocabulary that encourages viewing system behavior not in terms of calculations performed but in terms of rules applied. All of this has in turn led to a broader conception of the roles software can play.

A Broader View

Much software traditionally used for scientific and data processing applications has been based on well-established algorithms and has focused on performance, that is, finding the answer to the problem at hand. Knowledge-based systems have been applied to tasks like medical diagnosis, where few well-defined algorithms exist. As a result, significant emphasis is placed on the careful specification and accumulation of the knowledge necessary for the task. This in turn has focused interest on having the systems play a broader role, considering in addition to performance such capabilities as explanation, knowledge acquisition, and tutoring.

Many knowledge-based systems explain their results in the manner suggested earlier, keeping a record of the rules that have been applied and playing back the relevant portion of that record (*4*). Some systems are also able to answer questions about what they did not do (for example, "Why didn't you conclude that the organism was an *Escherichia coli*?"). The system examines the knowledge base to find all the rules that would have led to that conclusion, displaying them and showing why each failed to apply to the case at hand (*22*). General questions about the task (for example, "How do you conclude that the organism might be *E. coli*?) can be answered similarly, retrieving relevant rules from the knowledge base.

Knowledge acquisition is important because the accumulation of any sizable knowledge base is a process of iterative refinement, in part because the initial explication of previously tacit knowledge is difficult, often resulting in incomplete or inconsistent sets of rules. For example, a knowledge base might be judged incomplete if it contained rule 27 but no rules for inferring portal of entry, because it would be better for the system to infer the answer than to rely on the user's judgment. In response, one or more new rules might be added to allow the system to deduce the answer from more basic data. Now when the system, reasoning backward through the rules, comes to clause 3 of rule 27, rather than ask the user it will retrieve the newly added rules, which in turn may require the more basic observations. Adding new rules is rarely this straightforward in practice (*23*), but the process is facilitated by focusing the rules on the knowledge underlying the task.

Knowledge acquisition can also aid in the development and systematization of knowledge in a field. It assists in much the same way that writing a text often helps: building a system requires the same sort of methodical listing. Building a system helps in a way that developing any kind of computer program helps, and in a way that no paper listing can: it provides a form of "mental hygiene" in running the rules that is not available in other media. With rules written on paper and invoked by hand, there is enormous temptation to do what we think the rule meant, not what it actually said. Scale is also an issue: the ability to keep track of detail in hand simulation is easily overwhelmed by more than a few dozen rules. In that case the system can be used to do the reasoning and track the details.

The system also helps by running endless tests and running them faster than is possible by hand. This permits an interactive approach to knowledge acquisition: the system can be used to evaluate many different additions to the knowledge base. Such evaluation in turn facilitates incremental knowledge specification, accumulating bits of knowledge by distilling them one at a time from hundreds of real examples, an approach that has proven successful on tasks chosen in part because there is as yet no simple, encompassing theory for them.

Building a knowledge-based system also offers leverage because of its ability to provide explanations and because of its incremental character. Explanations ensure that we can see how the rules are actually being used, which may turn out to be different from what was intended.

Building a system incrementally makes it possible to attack a small part of a problem and, over time, add new knowledge to expand the scope of competence. As a result, it may not be necessary to understand the entire problem before starting on the parts that are understood.

Having laboriously transferred knowledge from an expert to the system, it would be useful if the system could transfer the same knowledge to a student. This transfer is possible but nontrivial (*24*), in part because teaching sometimes requires a deeper understanding than doing, and in part because it also requires pedagogical skills. Where problems can often be solved with the rote application of a procedure, teaching also requires being able to document why and how the procedure works, knowing how to structure a lesson, and so on.

Current Systems

While this broader view of programs has implications for the long term, much of the near-term impact of knowledge-based systems results from their performance on problems of pragmatic value. A number of systems are currently in use and under development in both the research and commercial communities. The representative systems described below were selected on the basis of applicability to a problem of scientific interest, substantial development, and current performance.

Geology. The Prospector system (*8*) was developed to aid in

mineral exploration, including regional resource evaluation, ore deposit identification, and drilling site selection. The system's knowledge base covered ten different kinds of deposits in detail, describing the characteristic features of large-scale deposits of lead, zinc, copper, molybdenum, uranium, and others. In one instance it was able to extend the boundaries of a known deposit (*25*).

The Dipmeter Advisor system (*26*) interprets data from oil wells. Tools lowered into the well and slowly retrieved measure such things as the inclination (or dip) of rock bedding planes. Using knowledge of the effects of geologic forces, human experts can interpret these measurements to infer subsurface geology, including the presence of faults, ancient sand dunes, river channels, and so forth. This knowledge was analyzed and embodied in the system, allowing it to produce interpretations, primarily for the deltaic environments in which it was originally developed.

Electronic troubleshooting. ACE (automated cable expert) was developed to analyze the large volume of trouble reports generated in residential telephone service (*27*). These reports were previously analyzed by looking for patterns of misbehavior that suggest a specific cable, junction box, or other component as a source of trouble. This expertise has been captured in approximately 500 rules that scan the trouble report database, processing overnight what had previously required a week or more.

Aeronautics—astronautics. The Navex system is designed to function as an interpreter of navigation data gathered during space shuttle flights (*9*). The shuttle is tracked by multiple radar sites, each providing estimates of range and azimuth. Like the human console operators, Navex determines whether to include or exclude specific tracking stations (on the basis of the quality of the data being supplied), whether to adjust (restart) the Kalman filter used to estimate position, velocity, and acceleration from the possibly noisy data, and whether to certify the current position estimate as reliable (for use by other groups at Mission Control).

The system also maintains a model of the mission, which, combined with its knowledge of the domain, allows it to anticipate events. As one example, the system knows when the solid rocket boosters are due to separate and knows that when they do, the C-band radars will often track them, rather than the shuttle.

Chemistry. The Dendral program (*28*), one of the earliest knowledge-based systems, aids in the determination of molecular structures by interpreting data from a mass spectrometer. Its expertise includes aliphatic structures like ketones, ethers, alcohols, and amines, and extends to cyclic ketones, estrogenic steroids, and prostaglandins. It has been used to check solutions found in the literature and often turns up additional possibilities; in one instance it proved better than human experts at untangling the data produced by a mixture of compounds.

Dendral's power arises from several kinds of knowledge of chemistry. Substantial performance arises from its structure generator, an exhaustive, nonredundant enumerator of possible structures that ensures no potential solution will be overlooked. Rules concerning structures that are impossible at room temperature provide additional knowledge used to guide and constrain the generator. Guidance is also provided by knowledge about mass spectra: various patterns of peaks are characteristic of particular families of molecules. Finally, knowledge of molecular fragmentation processes makes possible rules that test candidate solutions.

Medicine. Medicine has been a rich source of examples and challenging problems for these systems. Mycin (*6*) was designed to consult on problems of infectious disease diagnosis and therapy selection (diagnosing bacteremia and meningitis, then selecting antibiotic treatment). Caduceus (*7*) works in the broad problem of internal medicine; after almost 15 years of knowledge base construction effort the system is scheduled for field test late in 1986.

Mathematics. Macsyma (*10*), another of the earliest knowledge-based systems, can perform, at the user's request, a very wide range of operations in symbolic mathematics. Experimental systems have also been developed for statistics, where the interesting problem is not in doing the statistical calculations, but in knowing what arithmetic to do and how to interpret the results. Systems under development assist with determining when regression can be applied (*29*), basing their decision in part on knowledge of phenomena like collinearity.

The technology has been applied across a wide range of fields: in addition to the well-established systems mentioned, experimental systems have also been developed for tasks as diverse as planning experiments in molecular biology (*30*), designing organic syntheses (*31*), and investigating creativity in mathematical research (*32*).

Research Issues

While construction of systems like these has in a few cases been relatively straightforward (*15*), often the effort encounters substantial issues that are the subject of basic research. This section discusses a few representative issues.

Where it has been possible, distinguishing between what to know and what to do provides substantial benefits in terms of multiple uses of knowledge. But even under these circumstances, explanation and knowledge acquisition are not trivial. Explanation viewed as a replay of the reasoning is not particularly useful if that reasoning is long and complex. In response, some systems have modeled their tasks at multiple levels of detail and are thus capable of providing high-level explanations that avoid unnecessary detail (*33*). A key issue in research on automating knowledge acquisition is credit assignment, determining what missing or incorrect knowledge caused the system to produce the wrong answer. This in turn appears to require substantial knowledge about the task at hand (*34*), offering a technical version of the common-sense observation that learning proceeds best when it has an established base of knowledge to work from.

Experience has also indicated that it is difficult to keep this distinction sharp. Difficulties arise because an important part of expertise in many tasks is knowledge of sequence and problem-solving strategies. Xcon (*35*), for example, a large system with over 3000 rules, checks the purchase order for a computer system to ensure that it is complete (all of the appropriate components are present) and consistent. Six basic steps must be followed, in order, to accomplish this task, but there is no direct way to express this fact by using rules of the sort illustrated earlier. Such rules are well suited to expressing small, independent inferences, not the overall structure of the solution. Expressing overall solutions is not impossible through the use of rules, only very difficult and often indirect (*36*), intermixing the domain knowledge and how it is to be used, thereby losing clarity and the other advantages this distinction offers. While the sequencing of steps is easily expressed in any standard programming language, it is not yet obvious how to combine rules and procedures in the same program to get the benefits of both.

Problem-solving strategies express ways of attacking a problem, such as "use process of elimination," or "generate all hypotheses consistent with the initial data, then generate tests that will distinguish among hypotheses." They are ubiquitous in analytic problems like the diagnosis and data-interpretation tasks often encountered by knowledge-based systems. Yet neither rules of the sort illustrated earlier nor any of the standard knowledge representation techniques offer a way to express this information directly.

This difficulty has led to an active body of research whose fundamental hypothesis is that specifying and selecting a problem-

solving strategy—deciding what to do next—is itself a knowledge-based task (*37*). In response, strategies are treated as a task domain in much the same way that the original knowledge-based systems focused on their tasks: specify and accumulate a body of knowledge about the topic. This new body of knowledge is used to select a strategy appropriate to the problem at hand; that strategy is then used to guide application of knowledge in the original knowledge base. The difficult problems here are discovering the strategies and developing knowledge representation languages that are both expressive and efficient (*38*).

But even where distinction between knowing and doing has not been maintained with complete clarity, substantial benefit has resulted. Developing knowledge bases for a range of tasks has helped to make clear what knowledge is required for those problems. This in turn has made it possible to study the knowledge bases to detect recurring patterns of reasoning that might not otherwise have been visible (*39*). Ways of expressing those patterns now being developed should improve the performance and ease of construction of future systems.

A second area of considerable research activity is building systems that reason from a more fundamental understanding than is typically captured in rules. Traditionally rules have expressed empirical associations, links like those between symptoms and diseases where the connection has been observed frequently, but the underlying mechanism may not be clear. Rule 27, for instance, asserts such an association, but does not indicate why it is true, perhaps because the answer is not yet known.

Research efforts in progress draw on examples from analog and digital circuits, simple mechanical devices, and selected areas of medicine (*33, 40*). The key issue is understanding, modeling, and reasoning about causality: a central concept in many of the efforts is the notion of a causal explanation. While the equations for the force on a simple pendulum, for example, can be solved analytically to indicate that it will oscillate, this doesn't tell us what causes the oscillation. Physical intuition is better served by an explanation that describes how the forces at the beginning of the swing accelerate the bob from rest, how those forces then decrease as the string moves toward the vertical, then reverse, and so forth. The explanation is causal in that it identifies explicit causes and their effects, the causes precede effects, and effects propagate locally (there is no action at a distance)—all intuitive attributes of causality. The explanation is also qualitative, using terms like "increase" and "decrease," enabling it to suppress the detail present in a quantitative account.

Explanations like this play a key role in many forms of problem-solving currently being investigated. One interesting task is producing the explanations themselves: the system is given a structural description of a device and produces from it a causal description of the device's behavior (*41*). This differs from the task of traditional simulation, in which the basic question is what the device will do; here the goal is to produce an explanation of why such behavior occurs.

Such explanations can provide the basis for a powerful form of diagnostic reasoning (*42*). If, for example, a digital circuit is malfunctioning, tracing backward from a symptom along the path of causality permits the identification of components that may be broken. Systems built this way can handle a wider range of devices: where the knowledge base of a rule-based system is typically specific to a single device, programs built using this approach can in effect "read the schematics" to guide their reasoning. They are also more robust, in the sense that they can diagnose a wider range of faults, including those that change the connectivity of the circuit, changing the path of signal flow. To date only very simple circuits have been used; the difficult research problems include describing and reasoning about the behavior of complex devices like microprocessors.

In the long term this line of work has implications for both theory and practice. Theory is advanced by the careful statement of the principles that guide construction of these systems (*43*). One such principle, common in many engineering disciplines, indicates that the behavior specification of a component should refer only to properties of that component, not anything that component may be attached to. Thus the behavior of a switch, for instance, should not be "if the switch is closed, current will flow," since that presumes the switch is attached to a current source. Instead it should simply be "when the switch is closed, the voltage at either end of the switch will be the same." Following this guideline helps to ensure that the model captures physical causality, rather than the biases and expectations of the model builder.

Advances may lead to knowledge-based systems with a wider range of abilities. Current systems are narrowly focused in their expertise and are fragile, in the sense that their performance falls off precipitously near the boundaries of that expertise, especially when compared with the more graceful decline exhibited by human experts. These systems know the rules, but have no more fundamental theory to fall back on when no rules apply. Their rule base captures knowledge as a large collection of specific situations, but provides no means for reasoning that the current situation is different from "but very close to" one specified in a rule. Causal reasoning may provide a basis for such reasoning, broadening the scope of the system's performance.

Summary

Knowledge-based systems have generated wide interest for the kinds of problems they have attacked and the level of performance they have occasionally achieved. A more significant impact may be the broader view they take of a computer program, suggesting that it not be considered simply as a set of instructions written for the efficient solution of a problem, but viewed instead as an environment for the development, accumulation, and specification of knowledge about that problem.

This view is made possible in part by distinguishing between what to know and what to do, attempting as far as possible to express knowledge about the task without also committing to a specific way of using that knowledge. The separation of inference engine and knowledge base in these systems reflects this distinction; it also offers the opportunity to build new systems by writing new knowledge bases. Where the knowledge has successfully been expressed without committing to a specific way of using it, the same information can be used in multiple different ways, providing significant support for explanation, knowledge acquisition, and tutoring.

This distinction has been achieved to only a limited extent in most operational systems, but even so the systems built have proved to be useful tools aiding in the organization and study of knowledge about a range of tasks. These systems have also motivated research into the elucidation of strategies and the development of advanced languages for expressing them. Research focused on understanding and reasoning from causality emphasizes the accumulation of principles for careful construction of models and may provide an important base for broadening performance of these systems beyond their currently narrow focus.

While the computational technology described plays a key role in facilitating the elucidation, expression, and testing of knowledge, the specific computer programs built may prove to be irrelevant. The most lasting contribution of these systems will likely lie in the knowledge that had to be accumulated and formalized to make them work.

REFERENCES AND NOTES

1. A. Newell and H. A. Simon, *Human Problem Solving* (Prentice-Hall, Englewood Cliffs, NJ, 1972).
2. E. A. Feigenbaum, B. Buchanan, J. Lederberg, in *Machine Intelligence* B. Meltzer and D. Michie, Eds. (American Elsevier, New York, 1971), vol. 6.
3. The related term "expert system" is less desirable for its lack of technical substance and tendency to suggest inappropriate standards. Calling something an expert system primarily advertises the aspiration to have it perform at the level of human experts. Although this has been accomplished in several cases, significant utility has arisen from knowledge-based systems that function as intelligent assistants and colleagues, without ever becoming expert at their task.
4. R. O. Duda and E. H. Shortliffe, *Science* **220**, 261 (1983).
5. I coined the term "inference engine" in 1974 in conscious analogy to C. Babbage's term for his pioneering 19th-century machine, the "analytical engine," but with the appropriate modification to make clear that its basic operation is inference, not calculation.
6. B. G. Buchanan and E. H. Shortliffe, *Rule-Based Expert Systems* (Addison-Wesley, Reading, MA, 1984).
7. H. E. Pople, in *AI in Medicine*, P. Szolovits, Ed. (Westview, Boulder, CO, 1982).
8. R. O. Duda and R. Reboh, in *AI Applications for Business*, W. Reitman, Ed. (Ablex, Norwood, NJ, 1984).
9. M. C. Maletz, *Artificial Intelligence 1985* (1985), p. 71
10. R. Bogen *et al.*, "*MACSYMA Reference Manual*" (Laboratory for Computer Science report, Massachusetts Institute of Technology, Cambridge, 1975).
11. *Proceedings of the AAAI Workshop on Uncertainty and Probability in Artificial Intelligence* (American Association for Artificial Intelligence, Menlo Park, CA, 1985).
12. J. Fox, C. D. Meyers, M. F. Greaves, S. Pegram, *Methods Inform. Med.* **24**, 65 (1985).
13. This explanation has been simplified to present only the points needed for what follows. A more detailed explanation can be found in (*4*) or (*6*).
14. B. K. P. Horn, *Robot Vision* (MIT Press, Cambridge, MA, 1984).
15. J. Kunz *et al.*, "A physiological rule-based system for interpreting pulmonary function results" (Computer Science Department working paper HPP-78-19, Stanford University, Stanford, CA, 1978).
16. J. S. Bennett, L. A. Creary, R. S. Engelmore, R. E. Melosh, "SACON: A knowledge-based consultant in structural analysis" (Computer Science Department report HPP-78-23, Stanford University, Stanford, CA, 1978).
17. J. S. Bennett and C. R. Hollander, *Proc. Int. Joint Conf. AI* **7**, 243 (1981).
18. W. vanMelle, "A domain independent system that aids in constructing consultation programs" (Computer Science Department report STAN-CS-80-820, Stanford University, Stanford, CA, 1980).
19. The concept that programs might be given advice rather than instructions appears quite early in the history of AI [J. McCarthy, in *Semantic Information Processing*, M. Minsky, Ed. (MIT Press, Cambridge, MA, 1968)]. This same motivation is also at the heart of some of the work on languages like Prolog.
20. This is not always easy. (i) Each equation has to be inverted; this can be difficult in complex models. (ii) It may be necessary to keep track of alternative choices: if $c = a + b$, for instance, c can be changed by changing a alone, b alone, or both together. But often it can be done and in those situations illustrates an important technique. Even where equations cannot be inverted, there is still utility in moving backward through the model to determine which quantities are relevant to the desired result.
21. Various commercial spreadsheet programs have several of these capabilities; a small, experimental program used in the MIT course on knowledge-based systems does them all (crudely).
22. R. Davis and D. B. Lenat, *Knowledge-Based Systems in AI* (McGraw-Hill, New York, 1982).
23. J. Bachant and J. McDermott, *AI Mag.* **5**, 21 (1984); R. Davis, *Artif. Intell.* **12**, 121 (1979).
24. W. J. Clancey, *Int. J. Man-Mach. Stud.* **11**, 25 (1979).
25. A. N. Campbell, V. F. Hollister, R. O. Duda, P. E. Hart, *Science* **217**, 927 (1982).
26. R. G. Smith, *AI Mag.* **5**, 61 (1984).
27. G. T. Vesdoner, S. J. Stolfo, J. E. Zielinski, F. D. Miller, D. H. Copp, *Proc. Int. Conf. AI* **8**, 116 (1983).
28. R. K. Lindsay, B. G. Buchanan, E. A. Feigenbaum, J. Lederberg, *Applications of Artificial Intelligence for Organic Chemistry* (McGraw-Hill, New York, 1980).
29. R. W. Olford and S. C. Peters, in *Artificial Intelligence and Statistics*, D. Pregibon, Ed. (Addison-Wesley, Reading, MA, 1985); D. Pregibon, *ibid.*, p. 136.
30. M. Stefik, *Artif. Intell.* **16**, 111 (1981); P. Friedland and Y. Iwasaki, *J. Automated Reasoning* **1**, 161 (1985).
31. W. T. Wipke, H. Braun, G. Smith, F. Choplin, W. Sieber, in *Computer-Assisted Organic Synthesis*, W. T. Wipke and W. J. House, Eds. (American Chemical Society, Washington, DC, 1977).
32. D. B. Lenat, *Artif. Intell.* **9**, 257 (1977); D. B. Lenat and J. S. Brown, *ibid.* **23**, 269 (1984).
33. R. Patil, "Causal understanding of patient illness for electrolyte and acid-base diagnosis" (Computer Science Department, report MIT/LCS/TR-267, Massachusetts Institute of Technology, Cambridge, 1981).
34. J. G. Carbonell, R. S. Michalski, T. Mitchell, in *Machine Learning*, R. S. Michalski, J. G. Carbonell, T. Mitchell, Eds. (Tioga, Palo Alto, CA, 1983).
35. J. McDermott, *Artif. Intell.* **19**, 39 (1982).
36. C. Forgy and J. McDermott, *Proc. Int. Joint Conf. AI* **5**, 933 (1977).
37. R. Davis, *Artif. Intell.* **15**, 179 (1980); M. R. Genesereth, *Proc. Natl. Conf. AI* (1983), p. 119; W. J. Clancey, *ibid.*, p. 74.
38. H. J. Levesque, R. J. Brachman, in *Readings in Knowledge Representation*, R. J. Brachman and H. J. Levesque, Eds. (Morgan-Kaufmann, Los Altos, CA, 1985).
39. W. Clancey, *Artif. Intell.* **27**, 289 (1985).
40. *Artif. Intell.* **24**, 1–491 (1984).
41. J. deKleer, in (*40*), p. 205; A. Stevens *et al.*, "Steamer: advanced computer-aided instruction in propulsion engineering" (report 4702, Bolt, Beranek & Newman, Cambridge, MA, 1981).
42. R. Davis, in (*40*), p. 247; M. R. Genesereth, in (*40*), p. 411.
43. J. deKleer and J. S. Brown, in (*40*), p. 7.

44. Useful comments on earlier drafts were received from B. Williams, R. Duda, W. Hamscher, M. Shirley, D. Weld, T. Malone, R. Valdes-Perez, P. Szolovits, D. Lenat, and E. Feigenbaum.

Small Shared-Memory Multiprocessors

FOREST BASKETT AND JOHN L. HENNESSY

Multiprocessors built from today's microprocessors are economically attractive. Although we can use these multiprocessors for time-sharing applications, it would be preferable to use them as true parallel processors. One key to achieving efficient parallel processing is to match the communications capabilities of the multiprocessor to the communications needs of the problem. The other key is improved parallel programming systems. If these are achieved, then efficient parallel processing can be approached from both ends by providing more communications capability in the hardware and restructuring the problem to reduce the communications requirements.

THE LABORATORY COMPUTING ENVIRONMENT FOR SCIENtists and engineers has been changed dramatically, first by minicomputers and, more recently, by personal computers and powerful workstations. Small multiprocessors promise to increase the speed with which computationally intensive problems can be solved in the laboratory (*1*). While such machines should be as much as one order of magnitude faster than today's minicomputers, they will probably have comparable or lower costs. These multiprocessors will be small in that the processors will number on the order

Forest Baskett is director, Western Research Laboratory, Digital Equipment Corporation, 100 Hamilton Avenue, Palo Alto, CA 94301. John L. Hennessy is associate professor, Departments of Electrical Engineering and Computer Science, Stanford University, Stanford, CA 94305.