EXPERT
EDITORIAL

Remarks of a Retiring EIC

AI, Knowledge, and the Quest for Smart Systems

B. Chandrasekaran, Ohio State University

I have now served five years as the editor in chief of *IEEE Expert*. These have been important years for the magazine — I can honestly say that, with the help of a strong editorial board, *IEEE Expert* is now the preeminent international publication for intelligent system applications. The editorial board, the author list, and the subscription base are international, representing all the regions of the world where leading work in intelligent systems is carried on.

As EIC, I've had to think about the field in global and strategic terms so that the magazine could guide its readers to the most vital information. In this capacity, I've inevitably developed some opinions about both the field and how the advances in it are presented to readers. In the rest of this article, I offer some loosely connected remarks about AI and the technology of intelligent systems. These views have their origin at least partly in my role as the EIC of *IEEE Expert*.

The state of the field

AI is in creative turmoil. Although "expert systems" (or "knowledge-based systems") provided the impetus for the launching of this magazine nine years ago, other techniques for building intelligent systems have since been coming from other sources, such as neural nets, fuzzy sets, smart database technologies, learning methods, agent technologies, and robotics. Such new ideas have raised basic questions about how to think about intelligence, and have also made possible new technologies for solving some problems in AI. As a result, *IEEE Expert* has evolved into a forum for intelligent systems in general. Its logo redesign a couple of years ago emphasized the magazine's subtitle, and the breadth of its interest in intelligent systems.

In applications, AI has been finding its way into more and more tasks and domains, often in forms that we would be hard put to recognize as AI. For example, expert systems —once the focus of an unsustainable amount of interest — are now being integrated with more traditional technologies to solve a variety of problems. The convoluted processes by which organizations absorb expert systems technology would be a fascinating subject for study by future historians of technology. Quite apart from the technology itself, which has had an uneven ride, the idea of representing knowledge has raised research questions that have enormous potential for future computer systems.

There also continues to be philosophical interest in AI. Every 20 years or so, a new generation of philosophical critics discovers why we won't be able to create an artificial intelligence. Sometimes, it is a familiar reason: Godel's Theorem shows that machines can't be as smart as people, for example. Sometimes, it is a new reason: machines can appear to be smart, but they can't really understand. Of course, very little real AI work in the field depends on these issues being resolved one way or another. These issues do not have much to do with the creation of real-world systems that get increasingly smart in whatever sense of the term.

The quest for smart systems

Whatever the ups and downs of AI, the public's demand for smart systems has not slackened. The marketplace is full of products that are smart in some way: smart icons, wizards, intelligent support, intelligent access to data, and so on. Some of this stuff explicitly uses technology developed by the AI community, but much of it has no formal connection to AI. People in AI often respond to these products with disdain, if not horror, dismissing them as hype. This is a mistake. The technology of intelligent systems will continue to exist partly independent of AI as a field.

The public's definition of a smart product has nothing to do with a body of specific techniques, or even with the AI community's long-held goal of developing a general-purpose intelligence. The public does not expect a smart system to do everything that people do. It does expect a smart product to be flexible, adaptive, and robust. The product should "take some initiative," using background knowledge to handle unexpected situations so that the user need not "spell everything out" for the system. A product that has more of these qualities than its rivals is considered smarter.

Of course, after such a product has been around for a while, it will not seem as smart as newer products that better display these qualities. This commonsense notion about smartness is itself pretty smart. For one thing, it recognizes that smartness is an open-ended concept: products can get better or smarter in various ways, but there is no point past which a product is smart and

before which it is not. Secondly, there is no particular technology associated with smartness. Any product with extra power in one or more of these areas will be accepted as smart for a time, until a better rival comes along. Smartness is a fluid concept suggesting flexibility, generality, the possession of and ability to use knowledge, and in some situations, the ability to *acquire* knowledge, say, by learning.

There may well be nothing that sharply distinguishes technologies that help make smart products from those that help make unsmart ones. However, many in the AI community believe that the kind of computation that leads to smartness is clearly distinguishable from other kinds of computation or information processing. These researchers are often driven by the dream of making a general-purpose intelligent machine that can pass the Turing Test, or an autonomous robot that can move around the physical world, communicate with us in natural language, and use complex sensors to help it perceive the world much as we do. The machines in both these dreams share great generality and autonomy.

Such theoretical AI work will — in the long run — result in principles to help us build general, flexible, and autonomous products. But until then, a diverse body of AI and non-AI techniques will help us make many products smarter. In this sense, smart icons and presentation wizards are not simply marketing phrases to cash in on the public's naiveté, but in fact are perfectly reasonable, small-scale smart technologies.

If all this is true, then a magazine such as *IEEE Expert* has to take an increasingly broad view of intelligent systems and the techniques for constructing them. A system can be perceived as smart even if it lacks any recognizable AI techniques. It might draw its power mainly from the kinds of knowledge that it has, rather than from the complexity and depth of its runtime inference techniques. In fact, the knowledge might be used by garden-variety computing techniques.

AI as provider of ontology

Earlier, I mentioned that expert systems technology mutates in interesting ways as it makes its way through organizations. Consider what has happened to AI centers within large companies. A few years ago, these centers were veritable hives of knowledge engineering activity: knowledge representation tools were bought, knowledge encoded, search techniques implemented, and interesting systems built. Now, many of these companies no longer have central organizations devoted to AI technology.

When I ask the engineers how AI is faring in these companies, I usually get a story about how some system that was built for some operational division, using complex knowledge-engineering tools, was never really deployed. However, my informants go on, the operational people liked some of the things that the prototype did, and put their own people to work on understanding the prototype. In many cases, these studies identified useful combinations of algorithms that were then rebuilt and integrated seamlessly in the workaday computing environments of the operational world.

The public's definition of a smart product has nothing to do with a body of specific techniques, or even with the AI community's long-held goal of developing a general-purpose intelligence. The public does not expect a smart system to do everything that people do.

Typically, beneath all the rules and frames, there was a fairly simple, reliable algorithm, perhaps not quite as broad as the original system, but whose properties and behavior could be understood and reproduced. At that point, all the AI stuff seemed not only unnecessary, but positively in the way of making the algorithm work in the "real world." Or, if the heuristic parts were needed. they were hammered and polished into simple, understandable fragments that again could be made to work smoothly with existing operational codes using conventional languages. As a rule, the original AI idea was integrated and recoded into very traditional-looking software. The users never saw any AI, nor did anyone remember the system's origin in the AI lab, but the end result was a system that the user considered smart because of the amounts and types of knowledge it applied.

This scenario is a bit pat, but not by much. If most deployed AI systems go through such metamorphoses, what role do the AI ideas and tools play? Are they simply mistaken preludes to the correct solution? Or do they play a fundamental role?

Let's look closer at how the AI center constructed the prototype. As a rule, the task for which the prototype was built had not been identified as a problem that could be solved. For example, no one thought computers could be helpful for diagnostic tasks until expert systems research showed otherwise. Knowledge-based-systems technology was largely inspired by the deliberative aspect of intelligence studied by Allen Newell and Herbert Simon, where agents solve problems by using knowledge to set up and explore problem spaces. Thus, most early expert systems such as Mycin and R1 operated in problem spaces, even if they had enough knowledge to do little search. But there is another problem space search going on in the building of knowledge systems. The process of building the prototype (in the AI center) and then evolving it (in the operational division) until the core algorithm crystallized can itself be understood as a knowledge-driven search over a problem space of problems and solutions. That is, instead of creating an AI system to perform complex problem solving for the user, the AI lab used knowledge-engineering tools to search for interesting versions of the problem and core solutions for them. As the operational division recognized these, they were isolated and reimplemented with few, if any, components that explicitly operated in and explored problem spaces.

Thus, the AI center's work made two important contributions, even when the end product didn't use any of the AI techniques:

- It provided an *ontology*: a vocabulary for the task and the types of knowledge needed for the task. The ontology itself often required experimentation and analysis. Prototype systems in given domains had to be built, and their behavior understood and clarified, before the ontology needed could be determined.
- The process of building the prototype, experimenting with it, and modifying it revealed the *structure* of the problem and the solution. A version of the solution could then be implemented readily with traditional technologies.

The creators of knowledge system technology dreamed of thousands of machines,

each performing knowledge-based problem solving for users at runtime. Instead, the knowledge-engineering shells ended up being used in hundreds of AI labs to build prototypes to explore the nature of the tasks and to identify the ontologies and core problem-solution combinations.

When I talk to engineers in my sister academic disciplines, I continue to sense a great deal of enthusiasm for various AI ideas, such as functional reasoning, qualitative reasoning, and design problem solving. I also find that part of their enthusiasm stems from the fact that AI has identified new and interesting terms — ontologies — that they had been only informally aware of. For example, many engineers find the formal ontologies provided by qualitative physics and functional reasoning to be useful in their computational modeling, even when the systems they build don't use AI inference techniques.

I do not mean to imply that more complex AI technologies will not be deployed in major ways in the future. My point is that people often feel a product is smart simply because it seems to "know things." Much AI research has focused on what needs to be known for intelligent agents to perform various important and generic tasks. I expect that this kind of ontology creation — identification of knowledge types and how to use them — will continue to be an important contribution of AI research in the years to come.

Knowledge-level/symbol-level confusion

As EIC, I have read, or at least skimmed through, hundreds of papers over the past five years. I am surprised by the degree to which both researchers and practitioners fall in love with some general computational mechanism or another. Sometimes a mechanism is indeed wonderful, and without it some important problem may not be solved well or easily. Other times, however, the role of the mechanism itself is relatively unimportant. We need to separate the real reasons why a system works from the mechanism used to implement it.

In a previous article in this magazine, I analyzed this phenomenon by considering what I called the T-M-P triad: program P uses mechanism M to perform task T.[1] For example, a program might use fuzzy sets to solve some control task. Or a program such as Mycin might use a rule-based mechanism to solve a diagnostic task. Or a program could use PDP-style neural nets to perform visual word recognition.

Usually what happens is this: some individual or group becomes enthusiastic about a mechanism and, to show its power, builds a program that uses the mechanism to perform a task. Often the program works well, which is attributed to the mechanism.

Of course, with the same task T and mechanism M, two different people might write two different programs: P1, which works well, and P2, which doesn't. It is clear that each person added something different to M to make their program, and it is equally clear that what they added should get some of the credit for why P1 was successful and some of the blame for why P2 wasn't.

Sometimes, specific mechanisms are great because of how well they match the needs of the task. But in general, the fact that a task is successfully tackled by a program using a given mechanism is no reason to give much credit to the mechanism.

Conversely, the same person who built P1 using M would often be able to build another program P1′ for the same task using some other mechanism M′. For example, once we realize that the essence of Mycin's strategy is classification, we can use a frame language or Prolog to implement Mycin′ with essentially the same knowledge as Mycin and the same performance.

When M is computation-universal — when it can be used to build any computable function — then the relative contributions of M and the "something added" are even murkier. For example, suppose we use the mechanism of a Universal Turing Machine to write a language-understanding program; in some form or another, the program has to have theories of syntax, semantics and pragmatics, and the quality of the theories will have a strong effect on the program's success. The UTM should get some credit, of course, but most of us would be much more interested in the additional theories that the programmer used to make the program work.

For most computer scientists, all of this is pretty well known. Within AI, as well, there has been concern about misallocating credit to mechanisms at too low a level. Newell's knowledge level/symbol level proposal was an way of talking about programs in terms of the knowledge that can be attributed to them. In this view, a good deal of the power of a successful program is due to the knowledge in it.[2] David Marr proposed that there are three levels in explanation: an information processing strategy level, an algorithm level that implements the strategy, and a physical mechanism level that implements the algorithm.[3] He suggested that it is at the information processing strategy level that we should first seek the explanation of a complex computational phenomenon, such as biological vision. And in my own work on generic tasks, I have argued that we should understand successful knowledge-based systems in terms of the task-specific strategies they use, rather than the mechanisms — like frames or rules — in which they have been implemented.[4] The second-generation work on knowledge-based systems is based largely on this shift to the knowledge-level view of the field.

Sometimes, specific mechanisms are great because of how well they match the needs of the task. But in general, the fact that a task is successfully tackled by a program using a given mechanism is no reason to give much credit to the mechanism. Arguments have to be made about the role actually played by the mechanism. In spite of the generally increasing awareness of the knowledge-level view, I am still astonished at the degree to which articles submitted to *IEEE Expert* — actually, most papers in AI applications —are still oriented toward mechanisms, and only mechanisms. The articles begin with titles like "A Neural Net Approach to ..." or "A Fuzzy Set Approach to ...," and they proceed to describe the solution to some problem using their favorite mechanism. There is little discussion of whether the mechanism played the definitive role, whether some higher level content theory was the real reason for the success of the system, or even how one might tell. (For example, connectionist mechanisms sometimes contribute little to the success of a connectionist system.[5])

There are many reasons for this state of affairs. One is that the field is still in search

of a few magic mechanisms that explain intelligence. It is not widely appreciated that, even if the basic mechanisms are relatively few, complex behavior needs explanation at many levels of description. Of course, it is also true that the field has not developed a good vocabulary for talking about the higher level issues.

Another reason is that *technologies* as such are ultimately symbol-level things — they are mechanisms we compose to make machines that do things we want them to do. In our field, when we give someone a technology, we hand them a floppy disk (more likely, we point to the place on the WorldWideWeb from which they can download what would be on the disk) containing some symbol structures that interpret things written in some specific language: a mechanism. The state of AI technology is such that we are trying to work out a few simple, basic mechanisms and make them available for system builders. However much we analyze the sources of power as residing in higher level content theories, what people will recognize as technology are these mechanisms. Thus, it is understandable that there would be a heavy emphasis on low-level mechanisms.

All these reasons explain the fascination everyone has with mechanisms, but unless we keep an equal focus on content theories at many higher levels, and develop a sense of relative contributions of lower level mechanisms and higher level content theories in the success of the systems we build, then our ability to build — and teach how to build — complex intelligent systems will be limited.

In parting

It has been a sobering experience for me to receive letters regularly from readers in far corners of the world, expressing appreciation of some article or asking for further information about something that appeared in the magazine. I say "sobering" because such letters are reminders of the responsibilities that go with approving an article for publication in the magazine: the imprimatur of the IEEE and the Computer Society makes technical professionals all over the world expect accuracy and authority. The magazine is read, and people take the articles to represent the best of the technology. It has been a pleasure, and a source of pride, to have had the opportunity to lead *IEEE Expert* in the dissemination of the science and technology of intelligent systems for the past five years.

Acknowledgments

I thank John and Susan Josephson for useful comments on a draft of this article. With respect to my job as EIC, over the years, I have had the good fortune of working with two great managing editors: Henry Ayling and Steve Wilcox. Their associates, Dale Strok and Keri Schreiner, have also been great sources of help. I am thankful to all the members of the editorial board — a changing cast too numerous to mention individually — for their help in getting constructive and thorough reviews of papers submitted, and to all the guest editors for putting together a large number of topical special tracks. My thanks are due to the reviewers who are the real key to the quality of *IEEE Expert*. My secretary, Tanya Klenotic, has kept the article and correspondence flow smooth. Finally, I am thankful to Steve Cross, who agreed to take the job off my hands.

References

1. B. Chandrasekaran, "Broader Issues At Stake: A Response to Elkan," *IEEE Expert*, Vol. 9, No. 4, Aug. 1984, pp. 10–13.
2. A. Newell, "The Knowledge Level," *AI Magazine*, Summer 1981, pp. 1–19.
3. D. Marr, *Vision*, W.H. Freeman, New York, 1982.
4. B. Chandrasekaran, "Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design," *IEEE Expert*, Vol. 1, No. 3, Fall 1986, pp. 23–30.
5. B. Chandrasekaran, A. Goel, and D. Allemang, "Connectionism and Information Processing Abstractions: The Message Still Counts More than the Medium," *AI Magazine*, Vol. 9, No. 4, 1989, pp. 24–34.

B. Chandrasekaran is a professor of computer and information science at Ohio State University and has been editor in chief of *IEEE Expert* since January 1990. He can be reached at the Laboratory for AI Research, Ohio State Uni-versity, Columbus, OH 43210; Internet: chandra@cis.ohio-state.edu.