

Raj Reddy and Edward Feigenbaum have been seminal leaders in defining the emerging field of applied artificial intelligence and in demonstrating its technological significance. In 1994, they were named co-recipients of the A.M. Turing Award—ACM's most prestigious technical honor. We are pleased to present their insightful lectures that trace their professional development paths and expound on the roads ahead.

Edward A. Feigenbaum

How the “What” Becomes the “How”

IF I were a little older, I would have known Alan Turing. As it was, I missed him by a decade at the National Physical Laboratory.

I finished my Ph.D. work at Carnegie Tech (now Carnegie-Mellon University) and immediately shipped off to England on a Fulbright scholarship at the National Physical Laboratory (NPL). Turing had been there in the latter half of the 1940s, designing a computer, the ACE. In 1959, NPL scientists were still leisurely building onto the ACE. Many took long coffee breaks in the morning and long tea breaks in the afternoon, punctuated by the soldering in of a few transistors. Their spirited conversations often returned to Turing. For Ted Newman, Mike Woodger, and even Jim Wilkinson, Turing was literally an unforgettable presence. From these conversations of reminiscence, Turing became real for me, though he wasn't around any more.

Jim Wilkinson, in his 1970 Turing Award lecture [15], recalls the ACE computer project and the work he did with Turing on it. I saw Jim at his tireless labors, programming, debugging, and computing at the ACE's console. I even learned to program the ACE. After several decades, these memories of NPL and of Turing's ghostly presence are vivid.

I recently read the collection of Turing Award speeches from the first 20 years, and for the most part they are scholarly, technical, sober, as befits a solemn occasion. Except for the occasional paragraph, my predecessors carefully hid their joy. Not so for me. I want to express to all of you and to the ACM my thanks for the joy that you have given to me, to my family, to my Stanford colleagues and graduate students who have supported my work and my thoughts, and to the artificial intelligence (AI) research community.¹ It is a great honor to be

associated with Turing and all the previous recipients of this award.

Turing, the Visionary Genius

IN establishing our professional society's career research award, the fathers of the ACM chose Alan Turing as their icon because they viewed Turing as the father of modern computation. Only a few years after Turing's great theoretical work on computability, as World War II darkened Europe and threatened Britain, Turing was the intellectual leader of a group that designed and developed computers to crack the German U-boat codes. The Heath Robinson machine, perhaps the first high-speed electronic computer, led eventually to the building of the Colossus, an important, though secret, early computer.

In 1987, Hugh Whitmore wrote a play [14] based on Andrew Hodges' biography of Turing [5]. I remember vividly seeing this play in London. It was the most emotional experience I have ever had at the theater. I see and hear Turing delivering ideas to which I have dedicated my entire professional life and thought. I am transported back in time to visit with my intellectual grandparent, hearing him talk about what my work would be. Act II, Scene 1, a monologue; Turing is addressing the young boys at the Sherborne school, his old school, in 1953:

“People assume that computers are just glorified calculating machines. Not so. It's true that computers are often used to do calculating because they can calculate very quickly—but computer programs don't have to have anything to do with numbers. . . . A computer is a universal machine and I have proved how it can perform any task that can be described in

“Getting deeply involved in the world of a complex application is a voyage of discovery. The real world helps us to discover real and important issues, questions, and problems. Perhaps some of us could invent such insights, but I find that discovery is usually easier than invention.”

symbols. I would go further. It is my view that a computer can perform any task that the human brain can carry out. Any task. . . .”

Turing then speaks of learning programs and of chess-playing programs. He continues:

“The question thus arises as to whether or not we would credit such a machine with intelligence. I would say that we must. What I would very much like to do is to educate a computer, partly by direct training, partly by letting it find out things for itself. We don’t know how to do this yet, but I believe that it will be achieved in the very near future—and I feel sure that by the year 2,000, it will be considered perfectly correct to speak of an intelligent machine or to say that a computer is thinking. . . .”

To craft this monologue, Whitmore drew on writings and lectures of Turing that Hodges references in his biography.

Turing: The Engineer and the Applier

As a logician, Turing was a penetrating theorist, beyond perhaps all who have worked since. But we know him also, through his work on early computers and computational cryptography, as an applier, a builder of engineering prototypes, a designer who designed down to deep levels of detail.

This duality, which is common in physical science but not in computer science, enriched both aspects of Turing’s work, as it did mine. The citation for this Turing Award mentions the work of Reddy’s and mine in applied AI, but those applications are just one side of the coin. Tightly bound on the other are theoretical insights into the nature of knowledge-based systems, reasoning engines for using the knowledge bases, and even learning processes for automatic acquisition of new knowledge. In my view, there is no tension

between experimental science that exploits applications and theoretical science—between the empirical and the conceptual work.

Getting deeply involved in the world of a complex application is a voyage of discovery. The real world helps us to discover real and important issues, questions and problems. Perhaps some of us could invent such insights, but I find that discovery is usually easier than invention.

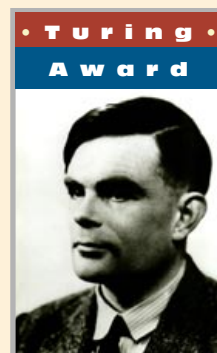
Turing Set the Terms of Reference for the AI Field

WITH his applied bent, and his work on real computers, Turing disliked meta-physical philosophizing about the vision of the intelligent machine. He preferred an operational definition of machine intelligence, and constructed one for us in his famous 1950 paper “Computing Machinery and Intelligence,” the so-called “Imitation Game” which we now call the “Turing Test” [13]. The Turing Test defines intelligence in terms of the flawless imitation of human cognitive behavior, i.e., thinking is what people do, and for a machine to be intelligent it must do what people do.

Sporadic experiments took place in those early years of the modern computing, but the first significant breakthrough came in 1956 with the construction of a heuristic program, the Logic Theory (LT) program, by Newell and Simon—one of the contributions for which they later won the Turing Award [11]. In their award lecture, they generalized the empirical results of the first part of the history of AI in a generalization that they named the “Physical Symbol System Hypothesis.” To paraphrase it colloquially, it says: “Yes, Turing’s vision was right. Programs for real digital computers can have the necessary and sufficient means for intelligent action.”

But what are these means? On this question, Turing was silent. Newell and Simon focused

¹Some of these people deserve special mention: Herbert Simon, my Ph.D. mentor and partner in my early EPAM work. My colleagues on the exciting journey of expert system innovation, Joshua Lederberg (who is the 1995 winner of the ACM Allen Newell Award) and Bruce Buchanan (the brilliant philosopher whose professional training I subverted to a better end). Three other AI colleagues have been important to my work and life: Peter Friedland (a master builder of applied AI systems), Raj Reddy (one of the great movers and shakers of AI), and my wife, Penny Nii, with whom I wrote programs, papers, and a book, and have shared a great life.



mainly on methods for problem solving and reasoning. The domain of work of LT was the propositional calculus, carefully chosen for this early voyage of discovery to decouple the issues of real-world semantics. The p 's and q 's of the propositional calculus are pure symbols. Newell and Simon's seminal paper on chess playing programs [10] is rich in ideas on problem solving but uses the word "knowledge" only once, and then only incidentally.

It took more than 10 years of additional experiments, primarily those for which my Stanford group is well known, to establish empirically that the means for intelligent action was primarily *knowledge*, and in most practical situations domain-specific knowledge [3]. We came to understand in a deep and technical way the challenge that Turing laid down in 1950.

What is AI Anyway?

Turing's operational test for intelligence views intelligence as most would view it, as something rather general-purpose—the ability to hold discourse about subjects that cover a broad range of human experience, using human language (natural language), but without using complex problem-solving skills to discover deep lines of reasoning. Turing's view was echoed in 1958 by McCarthy in his paper "Programs with Common Sense" [8], and more recently by Lenat and Guha in their papers on the CYC program [7].

But people are more than broad generalists with weak reasoning skills. Their most important intellectual behaviors—from the points of view of our culture, technology, and economy—are the behaviors of expertise, of specialists deeply trained and knowledgeable about the specifics of their various domains of work. A great physician or a great physicist is no more "generally" smart than you or I. He or she is smarter than we are about medicine and physics. To program a machine that would pass a Turing-like test in medicine or physics, the program would have to acquire, or be given, a great deal of domain-specific knowledge [3].

Is AI "Merely" Software? Yes, but Not "Merely"
Science works within frameworks of assumptions

that are usually not articulated. These assumptions constitute the *faith* of a science. Since Turing, the faith of AI has been that human intelligence is best modeled as software for a physical symbol system—the digital computer. That is, intelligent programs coexist with every other kind of software.

Actually, the programs of AI exist at one extreme of a software spectrum that I call the "What-to-How" spectrum of all software. (See accompanying figure.) The What end of the spectrum is the place at which people couple to computer systems to express what needs, desires, and goals they wish the computer to accomplish for them [3]. At the other extremum of the spectrum, the How end, is the computer hardware that can achieve those needs, desires, and goals in the extremely procedural step-by-step fashion that is the common stereotype of the computer field.

Computer science is largely a software field. The history of computer science has been a slow but steady traversal of the spectrum from the How end to the What end. Assemblers and compilers were early steps. What were called at an early time "higher-level languages"—Fortran and Cobol being the most popular examples—were big winners because they allowed users to express somewhat more easily the algebraic formulas and data-processing procedures the users wanted the computer to do for them. What the user wanted to say was easier to say.

In the same year that Fortran was born, AI programs were born at the What end of the spectrum. With these programs, users express their desires as goals—if not in full natural language then in the technical jargon of a specialty domain. Programs then use problem-solving processes and domain-specific knowledge to work out the details of how those goals will ultimately be accomplished entirely procedurally by the computer at the How end of the spectrum. Just as compilers translate expressions into executable code, so AI programs translate user goals into executable solution sequences.

Software technology has explored many other points on the spectrum, always striving away from the How toward the What. One small but important step was to push beyond algebraic formulas to the

*“We learn from Turing that envisioning is an important activity in science.
It sets research directions, leads us to important questions,
and motivates grandly.”*

particular array metaphor that is so much in use in daily human work—the spreadsheet. Come closer to the human need and—voilà!—instant success.

Recently there have grown (urgently, it seems) human needs and desires for intelligent aids for satisfying information goals, for searching over the vast Internet for specific information targets, or for searching through the immense complexities of desktop operating and application systems for the proper actions to accomplish a user desire. Thus the instant success of intelligent agent programs near the What end of the spectrum.

For reasons of brevity, I want to focus on one generalization and one additional location on the software spectrum. The former is about domain specificity, the latter about expert systems.

Domain-Specific Knowledge Is Critical for Mapping

THE path from the How of computer hardware to the What of human needs and desires is a path of increasing domain specificity.

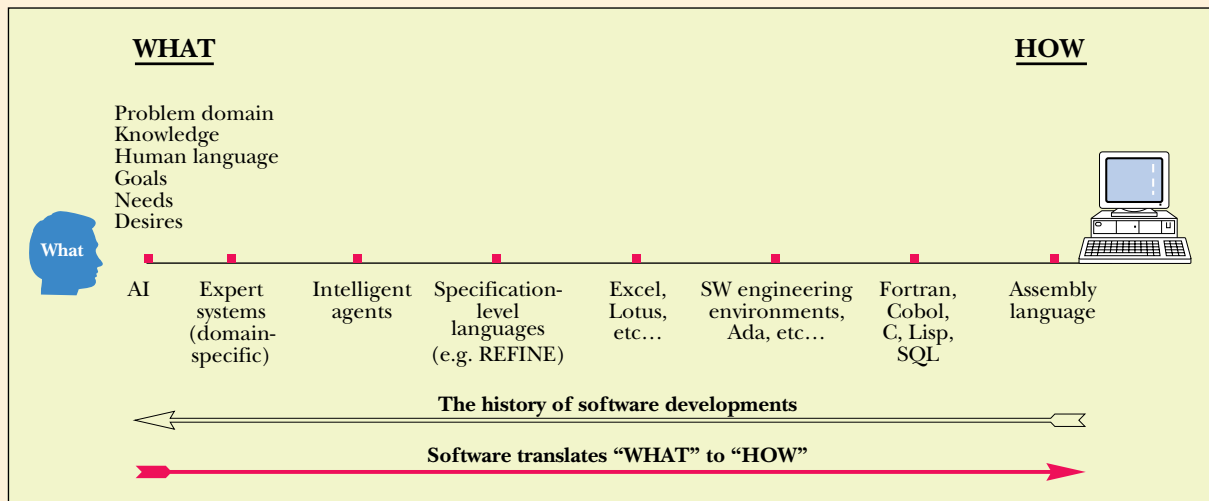
The AI extremum of the spectrum is no excep-

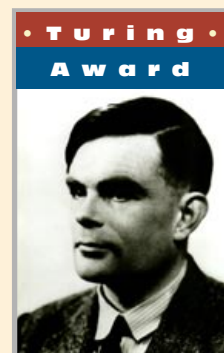
tion. The dialogues of Turing’s imitation game were general and conversational but were not about intellectually difficult real problems. Starting in 1965, Lederberg, Buchanan, and I began to attack such problems, first in chemistry, later in medicine, engineering, and some defense and industrial domains. As a result of these experiments, we were led to formulate a hypothesis about the power of domain-specific knowledge [3]. Similar experiments were done at CMU, Rutgers, and other places. Eventually a small expert systems software industry took shape, in which thousands of additional application experiments took place [6].

The results of all of these experiments was strong support for the view that in domain-specific knowledge lies the power of programs to solve problems. If the Physical Symbol System hypothesis is AI’s first and founding empirical law, then the Knowledge Principle (KP) is its second, and most important:

The power of AI programs to perform at high levels of competence is primarily a function of the pro-

The “What” to “How” spectrum of computing. Off to the left, the user wishes to express goals and needs in his or her own terms, not in terms of the computer’s step-by-step instructions. The history of the computer software is a series of steps that allow the user more of that freedom. AI software aspires to be at the point closest to the user’s desires and problems.





gram's knowledge of its task domain, and not of the program's reasoning processes.

Newell has labeled reasoning methods as “weak methods.” But they are not weak because they are poorly conceived. They are intrinsically weak until coupled to knowledge bases at the knowledge level (again, Newell’s term [9]).

In the knowledge lies the power.

At one level, the KP appears to state the obvious—as obvious as observing that physicians, not logicians, treat sick patients. But being obvious describes many important scientific generalizations, like Newton’s First and Third Laws or the Second Law of Thermodynamics.

At a deeper, technical level, the KP allows us to properly focus, prioritize, and evaluate AI research efforts and to plan and execute real-world applications of AI. Here are two examples:

- The Japanese national Fifth Generation (5G) Project gave highest priority, hence the most resources, to the reasoning process, developing ultrafast computers (How) for logic programs. The building of application knowledge bases (What) was treated almost as an afterthought. Hence the 5G computers solved few real problems.
- Years after the KP was formulated, it is now well understood that efforts toward Natural Language Understanding by computers should not focus on syntax-oriented approaches but rather focus on the knowledge bases needed to support understanding [7].

Minsky, in his 1969 Turing Award lecture, “Form and Content in Computer Science,” said :

“There is a real conflict between the logician’s goal and the educator’s. The logician wants to minimize the variety of ideas, and doesn’t mind a long thin path. The educator (rightly) wants to make the paths short and doesn’t mind—in fact, prefers—connections to many other ideas. . . .”

What Turing failed to tell us, for perhaps he thought we would intuitively understand it, is that common sense is mostly a knowledge-based game, not a logic-based game.

The KP Applies to the What-to-How Spectrum

Computer science textbooks give one the impression that software is about algorithms and generalized procedures. But that is an academic simplification and sterilization. The software that satisfies the real needs of real-world users is an artifact of details and exceptions. As Strassmann has noted [12], software is the technology of exceptions. To put it in my terms, both the power of software and the difficulty of constructing it lie in domain specificity.

The How end of the spectrum is extremely general—like a universal Turing machine, and almost as difficult to program. Fortran, and Cobol, among others, are somewhat more domain specific but also very general, hence one of the important elements of the perennial so-called “software crisis.” Spreadsheets are only slightly better, but in fact most of the programming done in these languages is done with domain-specific templates by ordinary folks who would be baffled if we called them programmers.

The hottest research topic these days among software specialists in computer science is domain-specific software architectures, another way of describing reuse at a point very close to domain details. When you look into the research projects, as one might have expected, they are all about architectures and not about domain specificity. Shades of the Japan’s 5G Project! Most computer scientists are uncomfortable with the specifics of domains. That is “someone else’s job,” right? Wrong. The domain-specific knowledge is almost all there is. If computer scientists distance themselves from domain-specific details, they are distancing themselves from relevance to the real software world. A key task for computer scientists is to design tools that allow users to express and capture domain-specific concepts and details, that represent these appropriately, and that specify architectures and processes using this knowledge.

Most of the best selling software packages are domain specific. Word processors are not general purpose systems; they are specific to text manipulation, formatting, and document preparation details. The presentation programs are specific to the details of bullet charts, builds, slide shows, and the like. The tax programs we use are specific to the

“The software that satisfies the real needs of real-world users is an artifact of details and exceptions. To put it in my terms, both the power of software and the difficulty of constructing it lie in domain specificity.”

forms and formulas of the Internal Revenue Service and to the specific tax advice we need to stay out of trouble while minimizing our tax.

Expert Systems: The KP in Center Stage

EARLIER, I proposed a view somewhat different from Turing's of intelligent computer programs. It is a view based on performance at high levels of competence on problems of intellectual difficulty, the kinds of problems on which experts display their expertise. In the 1960s, our Stanford group began calling such programs “expert systems (ES).” Because they achieved their performance power from domain-specific knowledge of particular application domains, they were not only landmark programs in AI but also instant applications. Since the 1970s, the class of ESs has become the largest and best known class of AI applications. Indeed, most of these applications are no longer called ESs, but have ascended to anonymity within the mainstream of computer software (akin to the Good ascending to heaven), where they acquire user-oriented names like business logic, tip wizards, and configurators.

An ES is essentially a symbolic computer model of human expertise in a specific domain of work. Thus an ES begins life with the dominant genes of its domain-specific nature.

The knowledge base (KB) of the ES contains the up-to-date knowledge and the informal heuristic know-how of one or more experts in the domain. The knowledge in the KB is what differentiates these experts from novices, even though all may employ the same logical reasoning methods.

The centrality of the KB, the focal point of an ES's domain specificity, tells us that the central technology of ES work is knowledge representation technology. Knowledge capturing (or knowledge acquisition) is actually equally important but is not yet much of a technology.

So it is no technological accident that AI and ES scientists helped to pioneer important data representation techniques in the software world. Objects and the various (and complex) forms of inheritance of properties are perhaps the most widespread. Also important are the rule-based representations, including the writing of rule

expressions in logic (such as business logic). Software companies, which at an early stage of their lives integrated these knowledge-representation techniques with reasoning processes, are now a major part of the software market for object-oriented application development tools. Their success signals that software developers in general have begun to realize the importance of domain-specific knowledge and are buying the tools needed to support its representation and management.

Expert System Applications

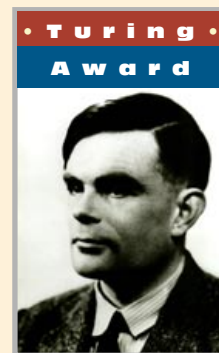
THE needs of literally millions of users at home and in commerce, industry, and defense are being served by ESs today. Tens of thousands of ESs were built worldwide during the first 15 years of commercialization, and many thousands are presently operational [4].

Ordinary people encounter expert systems embedded within best-seller software packages. The package “Tax Cut,” with cumulative sales of more than a million units, contains a rule-based tax advisor of a thousand rules, updated by a small staff of tax lawyers. Microsoft has begun to embed ESs that offer users tips and that diagnose and treat user needs for help. Trilogy, whose ads claim that it was the fastest-growing software company in the U.S., sells salesperson's aids for configuring equipment that is bought semi-custom, a type of application pioneered by Digital Equipment Corp. The KBs are prepared on contract by Trilogy, by third-party systems houses, or by users themselves.

Because organizations reduce risks by moving slowly into new technologies, following successful pioneers, the front of technology penetration is serrated, with many niche penetrations. These are the most significant ES niche areas to date:

Diagnosis of system failures or equipment failures. The users are in system-monitoring rooms, on shop floors, or at help desks helping customers. The monitoring and diagnosing of communication networks and manufacturing processes are important subsets.

Financial analysis and decision aids for transactions.



Examples include foreign exchange, bank loan-making, insurance underwriting, auditing, and tax. *Scheduling and planning of operations, manufacturing, logistics, and similar problems that are subject to a large number of complex interlocking constraints.* NASA's use in Space Shuttle refurbishing operations is an oft-cited landmark.

Configuration of equipment that is ordered and manufactured in a semi-custom manner. This is a critical technology for what is becoming known as "mass customization" in the modern economy.

Regulation helper systems provide expert guidance to users—citizens, government workers, or company employees—who have to solve problems in and around large bodies of rules and regulations that are almost too complex to comprehend. Employment law, pension systems, welfare rules, and tax codes are examples. Since the entry point to these is often forms, the expert system can serve as an intelligent form that watches and guides the form-filling user. Tax Cut, for example, offers advice in the context of the Form 1040 that you are filling out.

Measured by the usual metrics of corporate management, the economic leverage of deploying expertise as ES applications has been remarkably large. Returns on investment of hundreds of percent per year have been routinely reported. Similarly, payback periods of months, not years, are common. Often, millions of dollars per year are saved because of enhanced quality of decisions made. Not only cost savings but competitive edge over an opponent has been achieved because of the more rapid speed of decision making. For example, planning cycles that take minutes instead of hours, or hours instead of days, have helped NASA, the Defense Department in its various missions, and corporate manufacturing operations [4].

Why is a deployed computer model of expertise such a powerful economic lever?

Perhaps the answer is to be found in a book called *Augustine's Laws* by the CEO of the Lockheed Martin Corporation [1]. The laws are generalizations from stories of engineering

management of large aerospace projects. One of Augustine's laws, which I like to call the "power of expertise" law, plots a function of the measured performance on a particular task vs the number of people in the organization who achieve that level of performance. The curve is sharply skewed. Most of the high-level performance is performed by very few of the people. The best performers on the curve often outperform the worst performers by at least a factor of 10.

An order of magnitude of productivity is a big deal. If using ESs to deploy the expertise of the best to the computer consoles of the mediocre can raise the average performance level even a fraction of that factor of 10, then the economic leverage of the deployment will be huge.

Turing: Closing the Circle

CLOSE this lecture as I began, by thinking of Alan Turing, reflecting on his contribution to AI and computer science and on the future of these fields.

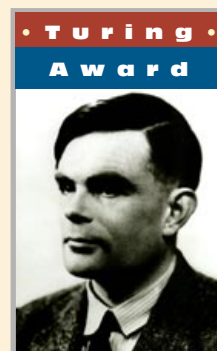
The age of electronic computers had barely dawned when Turing boldly and brilliantly envisioned their role as the medium for computational models of thought and their use as artifacts with intelligence. In the several places in which Turing's famous paper on machine intelligence [13] is reprinted, no editor has ever labeled it science fiction.

We learn from Turing that envisioning is an important activity in science. It sets research directions, leads us to important questions, and motivates grandly. I offer to you the advice given long ago by an American missionary to the young boys in Hokkaido, Japan: "Boys, be bold!"

In Arthur Clarke's book on envisioning, *Profiles of the Future* [2], there is yet another law of expertise.

Clarke's Law: "When a distinguished but elderly scientist states that something is possible, he is almost certainly right. When he states that something is impossible, he is very probably wrong."

According to his (then young) colleague, Donald Michie, Turing discussed concepts of machine



intelligence toward the end of World War II. After WW II, according to Hodges [5], in promoting his project to build the ACE computer, he stated audaciously that the real reason for building such a machine was to further experiments on machine intelligence. It was not a vision that would endear him to the government bureaucrats in charge of the decision whether to fund the ACE or not. Hodges quotes from a letter Turing wrote to Ross Ashby, the neuroscientist and physician turned cyberneticist: "In working on the ACE, I am more interested in the possibility of producing models of the action of the brain than in the practical applications to computing" [5, p. 363].

The vision of computational models of intelligence, to which we regularly apply (and then unapply) transient and trendy labels, is one of the great goals that science has envisioned.


It is a truly grand challenge for computer science.

Modeling thought is on the same grand scale as modeling the evolution of the universe from the Big Bang, or modeling the development of life from DNA and proteins.

The End

Breaking the Code, final scene, from Turing's last speech before taking his own life:

"What is needed is the ability to take ideas seriously and to follow them through to their logical if upsetting conclusion. . . . Can a mind exist without the body? Can mental processes take place in something other than a living brain? How are we to answer that question satisfactorily? Is it possible to do so? Or is it simply an eternal Entscheidungsproblem? Forever undecidable. . . . Being a practical man as well as a theorist, I tend to look for practical solutions. . . ."

Thank you, Turing, for your enduring legacy. 

References

1. Augustine, N.R. *Augustine's Laws*. Penguin, New York, 1987.
2. Clarke, A. *Profiles of the Future*. Bantam, New York, 1965.

3. Feigenbaum, E.A. The art of artificial intelligence: Themes and case studies of knowledge engineering. In *Proceedings of the International Joint Conference on Artificial Intelligence V* (Boston, 1977).
4. Feigenbaum, E.A., McCorduck, P., and Nii, H.P. *The Rise of the Expert Company*. Times Books, New York, 1988.
5. Hodges, A. *Alan Turing: The Enigma*. Vintage, 1992.
6. *Innovative Applications of Artificial Intelligence 1*, 6 AAAI Press, Menlo Park, Calif., 1989-94.
7. Lenat, D. Artificial intelligence. *Sci. Am.* 273, 3 (Sept. 1995), 80-82.
8. McCarthy, J. *Programs with Common Sense: Mechanization of Thought Processes, Vol. 1*. HMSO, London, 1959.
9. Newell, A. The knowledge level. *Artif. Intell.* 18 1982, 87-127.
10. Newell, A., Shaw, J.C., and Simon, H.A. Chess-playing programs and the problem of complexity. In Feigenbaum and Feldman, Eds., *Computer and Thought*, AAAI Press and MIT Press, Cambridge, Mass., 1995, pp. 39-70.
11. Newell, A., Shaw, J.C. and Simon, H.A. Empirical Explorations with the Logic Theory Machine, in Feigenbaum and Feldman, Eds. *Computer and Thought*, AAAI Press and MIT Press, Cambridge, Mass., 1995, pp. 109-133.
12. Strassman, P. *The Politics of Information Management*. 1994.
13. Turing, A. M. Computing machinery and intelligence. In E. Feigenbaum and J. Feldman, Eds. *Computers and Thought*, AAAI Press and MIT Press, Cambridge, Mass., 1995, pp. 11-35.
14. Whitmore, H. *Breaking the Code*. Samuel French, London, 1987.
15. Wilkinson, J.H. Some comments from a numerical analyst. In *ACM Turing Award Lectures: The First Twenty Years, 1966-1985*, R. Ashenurst and S. Graham, Eds. ACM Press, New York, and Addison-Wesley, Reading, Mass., 1987, pp. 243-256.

Edward A. Feigenbaum is Kumagai Professor of Computer Science, Stanford University; and Chief Scientist, United States Air Force. His present mailing address is Computer Science Department, Gates Hall, Stanford University, Stanford, CA 94305. email: feigenbaum@cs.stanford.edu.

Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© ACM 0002-0782/96/0500 \$3.50