1975 ACM Turing
Award Lecture

The 1975 ACM Turing Award was presented jointly to Allen Newell and Herbert A. Simon at the ACM Annual Conference in Minneapolis, October 20. In introducing the recipients, Bernard A. Galler, Chairman of the Turing Award Committee, read the following citation:

"It is a privilege to be able to present the ACM Turing Award to two friends of long standing, Professors Allen Newell and Herbert A. Simon, both of Carnegie-Mellon University.

"In joint scientific efforts extending over twenty years, initially in collaboration with J.C. Shaw at the RAND Corporation, and subsequently with numerous faculty and student colleagues at Carnegie-Mellon University, they have made basic contributions to artificial intelligence, the psychology of human cognition, and list processing.

"In artificial intelligence, they contributed to the establishment of the field as an area of scientific endeavor, to the development of heuristic programming generally, and of heuristic search, means-ends analysis, and methods of induction, in particular; providing demonstrations of the sufficiency of these mechanisms to solve interesting problems.

"In psychology, they were principal instigators of the idea that human cognition can be described in terms of a symbol system, and they have developed detailed theories for human problem solving, verbal learning and inductive behavior in a number of task domains, using computer programs embodying these theories to simulate the human behavior.

"They were apparently the inventors of list processing, and have been major contributors to both software technology and the development of the concept of the computer as a system of manipulating symbolic structures and not just as a processor of numerical data.

"It is an honor for Professors Newell and Simon to be given this award, but it is also an honor for ACM to be able to add their names to our list of recipients, since by their presence, they will add to the prestige and importance of the ACM Turing Award."

Computer Science as Empirical Inquiry: Symbols and Search

Allen Newell and Herbert A. Simon

Computer science is the study of the phenomena surrounding computers. The founders of this society understood this very well when they called themselves the Association for Computing Machinery. The machine—not just the hardware, but the programmed, living machine—is the organism we study.

This is the tenth Turing Lecture. The nine persons who preceded us on this platform have presented nine different views of computer science. For our organism, the machine, can be studied at many levels and from many sides. We are deeply honored to appear here today and to present yet another view, the one that has permeated the scientific work for which we have been

Key Words and Phrases: symbols, search, science, computer science, empirical, Turing, artificial intelligence, intelligence, list processing, cognition, heuristics, problem solving.

CR Categories: 1.0, 2.1, 3.3, 3.6, 5.7.

The authors' research over the years has been supported in part by the Advanced Research Projects Agency of the Department of Defense (monitored by the Air Force Office of Scientific Research) and in part by the National Institutes of Mental Health.

Authors' address: Carnegie-Mellon University, Pittsburgh.

cited. We wish to speak of computer science as empirical inquiry.

Our view is only one of many; the previous lectures make that clear. However, even taken together the lectures fail to cover the whole scope of our science. Many fundamental aspects of it have not been represented in these ten awards. And if the time ever arrives, surely not soon, when the compass has been boxed, when computer science has been discussed from every side, it will be time to start the cycle again. For the hare as lecturer will have to make an annual sprint to overtake the cumulation of small, incremental gains that the tortoise of scientific and technical development has achieved in his steady march. Each year will create a new gap and call for a new sprint, for in science there is no final word.

Computer science is an empirical discipline. We would have called it an experimental science, but like astronomy, economics, and geology, some of its unique forms of observation and experience do not fit a narrow stereotype of the experimental method. None the less, they are experiments. Each new machine that is built is an experiment. Actually constructing the machine poses a question to nature; and we listen for the answer by observing the machine in operation and analyzing it by all analytical and measurement means available. Each new program that is built is an expermient. It poses a question to nature, and its behavior offers clues to an answer. Neither machines nor programs are black boxes; they are artifacts that have been designed, both hardware and software, and we can open them up and look inside. We can relate their structure to their behavior and draw many lessons from a single experiment. We don't have to build 100 copies of, say, a theorem prover, to demonstrate statistically that it has not overcome the combinatorial explosion of search in the way hoped for. Inspection of the program in the light of a few runs reveals the flaw and lets us proceed to the next attempt.

We build computers and programs for many reasons. We build them to serve society and as tools for carrying out the economic tasks of society. But as basic scientists we build machines and programs as a way of discovering new phenomena and analyzing phenomena we already know about. Society often becomes confused about this, believing that computers and programs are to be constructed only for the economic use that can be made of them (or as intermediate items in a developmental sequence leading to such use). It needs to understand that the phenomena surrounding computers are deep and obscure, requiring much experimentation to assess their nature. It needs to understand that, as in any science, the gains that accrue from such experimentation and understanding pay off in the permanent acquisition of new techniques; and that it is these techniques that will create the instruments to help society in achieving its goals.

Our purpose here, however, is not to plead for understanding from an outside world. It is to examine one aspect of our science, the development of new basic understanding by empirical inquiry. This is best done by illustrations. We will be pardoned if, presuming upon the occasion, we choose our examples from the area of our own research. As will become apparent, these examples involve the whole development of artificial intelligence, especially in its early years. They rest on much more than our own personal contributions. And even where we have made direct contributions, this has been done in cooperation with others. Our collaborators have included especially Cliff Shaw, with whom we formed a team of three through the exciting period of the late fifties. But we have also worked with a great many colleagues and students at Carnegie-Mellon University.

Time permits taking up just two examples. The first is the development of the notion of a symbolic system. The second is the development of the notion of heuristic search. Both conceptions have deep significance for understanding how information is processed and how intelligence is achieved. However, they do not come close to exhausting the full scope of artificial intelligence, though they seem to us to be useful for exhibiting the nature of fundamental knowledge in this part of computer science.

I. Symbols and Physical Symbol Systems

One of the fundamental contributions to knowledge of computer science has been to explain, at a rather basic level, what symbols are. This explanation is a scientific proposition about Nature. It is empirically derived, with a long and gradual development.

Symbols lie at the root of intelligent action, which is, of course, the primary topic of artificial intelligence. For that matter, it is a primary question for all of computer science. For all information is processed by computers in the service of ends, and we measure the intelligence of a system by its ability to achieve stated ends in the face of variations, difficulties and complexities posed by the task environment. This general investment of computer science in attaining intelligence is obscured when the tasks being accomplished are

limited in scope, for then the full variations in the environment can be accurately foreseen. It becomes more obvious as we extend computers to more global, complex and knowledge-intensive tasks—as we attempt to make them our agents, capable of handling on their own the full contingencies of the natural world.

Our understanding of the systems requirements for intelligent action emerges slowly. It is composite, for no single elementary thing accounts for intelligence in all its manifestations. There is no "intelligence principle," just as there is no "vital principle" that conveys by its very nature the essence of life. But the lack of a simple *deus ex machina* does not imply that there are no structural requirements for intelligence. One such requirement is the ability to store and manipulate symbols. To put the scientific question, we may paraphrase the title of a famous paper by Warren McCulloch [1961]: What is a symbol, that intelligence may use it, and intelligence, that it may use a symbol?

Laws of Qualitative Structure

All sciences characterize the essential nature of the systems they study. These characterizations are invariably qualitative in nature, for they set the terms within which more detailed knowledge can be developed. Their essence can often be captured in very short, very general statements. One might judge these general laws, due to their limited specificity, as making relatively little contribution to the sum of a science, were it not for the historical evidence that shows them to be results of the greatest importance.

The Cell Doctrine in Biology. A good example of a law of qualitative structure is the cell doctrine in biology, which states that the basic building block of all living organisms is the cell. Cells come in a large variety of forms, though they all have a nucleus surrounded by protoplasm, the whole encased by a membrane. But this internal structure was not, historically, part of the specification of the cell doctrine; it was subsequent specificity developed by intensive investigation. The cell doctrine can be conveyed almost entirely by the statement we gave above, along with some vague notions about what size a cell can be. The impact of this law on biology, however, has been tremendous, and the lost motion in the field prior to its gradual acceptance was considerable.

Plate Tectonics in Geology. Geology provides an interesting example of a qualitative structure law, interesting because it has gained acceptance in the last decade and so its rise in status is still fresh in memory. The theory of plate tectonics asserts that the surface of the globe is a collection of huge plates—a few dozen in all—which move (at geological speeds) against, over, and under each other into the center of the earth, where they lose their identity. The movements of the plates account for the shapes and relative locations of the continents and oceans, for the areas of volcanic and earthquake activity, for the deep sea ridges, and so on. With a few additional particulars as to speed and size, the essential theory has been specified. It was of course not accepted until it succeeded in explaining a number of details, all of which hung together (e.g. accounting for flora, fauna, and stratification agreements between West Africa and Northeast South America). The plate tectonics theory is highly qualitative. Now that it is accepted, the whole earth seems to offer evidence for it everywhere, for we see the world in its terms.

The Germ Theory of Disease. It is little more than a century since Pasteur enunciated the germ theory of disease, a law of qualitative structure that produced a revolution in medicine. The theory proposes that most diseases are caused by the presence and multiplication in the body of tiny single-celled living organisms, and that contagion consists in the transmission of these organisms from one host to another. A large part of the elaboration of the theory consisted in identifying the organisms associated with specific diseases, describing them, and tracing their life histories. The fact that the law has many exceptions—that many diseases are not produced by germs—does not detract from its importance. The law tells us to look for a particular kind of cause; it does not insist that we will always find it.

The Doctrine of Atomism. The doctrine of atomism offers an interesting contrast to the three laws of qualitative structure we have just described. As it emerged from the work of Dalton and his demonstrations that the chemicals combined in fixed proportions, the law provided a typical example of qualitative structure: the elements are composed of small, uniform particles, differing from one element to another. But because the underlying species of atoms are so simple and limited in their variety, quantitative theories were soon formulated which assimilated all the general structure in the original qualitative hypothesis. With cells, tectonic plates, and germs, the variety of structure is so great that the underlying qualitative principle remains distinct, and its contribution to the total theory clearly discernible.

Conclusion. Laws of qualitative structure are seen everywhere in science. Some of our greatest scientific discoveries are to be found among them. As the examples illustrate, they often set the terms on which a whole science operates.

Physical Symbol Systems

Let us return to the topic of symbols, and define a *physical symbol system*. The adjective "physical" denotes two important features: (1) Such systems clearly obey the laws of physics—they are realizable by engineered systems made of engineered components; (2) although our use of the term "symbol" prefigures our intended interpretation, it is not restricted to human symbol systems.

A physical symbol system consists of a set of entities, called symbols, which are physical patterns that can occur as components of another type of entity called an expression (or symbol structure). Thus, a symbol structure is composed of a number of instances (or tokens) of symbols related in some physical way (such as one token being next to another). At any instant of time the system will contain a collection of these symbol structures. Besides these structures, the system also contains a collection of processes that operate on expressions to produce other expressions: processes of creation, modification, reproduction and destruction. A physical symbol system is a machine that produces through time an evolving collection of symbol structures. Such a system exists in a world of objects wider than just these symbolic expressions themselves.

Two notions are central to this structure of expressions, symbols, and objects: designation and interpretation.

> *Designation.* An expression designates an object if, given the expression, the system can either affect the object itself or behave in ways dependent on the object.

In either case, access to the object via the expression has been obtained, which is the essence of designation.

> *Interpretation.* The system can interpret an expression if the expression designates a process and if, given the expression, the system can carry out the process.

Interpretation implies a special form of dependent action: given an expression the system can perform the indicated process, which is to say, it can evoke and execute its own processes from expressions that designate them.

A system capable of designation and interpretation, in the sense just indicated, must also meet a number of additional requirements, of completeness and closure. We will have space only to mention these briefly; all of them are important and have far-reaching consequences.

(1) A symbol may be used to designate any expression whatsoever. That is, given a symbol, it is not prescribed a priori what expressions it can designate. This arbitrariness pertains only to symbols; the symbol tokens and their mutual relations determine what object is designated by a complex expression. (2) There exist expressions that designate every process of which the machine is capable. (3) There exist processes for creating any expression and for modifying any expression in arbitrary ways. (4) Expressions are stable; once created they will continue to exist until explicitly modified or deleted. (5) The number of expressions that the system can hold is essentially unbounded.

The type of system we have just defined is not unfamiliar to computer scientists. It bears a strong family resemblance to all general purpose computers. If a symbol manipulation language, such as LISP, is taken as defining a machine, then the kinship becomes truly brotherly. Our intent in laying out such a system is not to propose something new. Just the opposite: it is to show what is now known and hypothesized about systems that satisfy such a characterization.

We can now state a general scientific hypothesis—a law of qualitative structure for symbol systems:

> *The Physical Symbol System Hypothesis.* A physical symbol system has the necessary and sufficient means for general intelligent action.

By "necessary" we mean that any system that exhibits general intelligence will prove upon analysis to be a physical symbol system. By "sufficient" we mean that any physical symbol system of sufficient size can be organized further to exhibit general intelligence. By "general intelligent action" we wish to indicate the same scope of intelligence as we see in humian action: that in any real situation behavior approprate to the ends of the system and adaptive to the demands of the environment can occur, within some limits of speed and complexity.

The Physical Symbol System Hypothesis clearly is a law of qualitative structure. It specifies a general class of systems within which one will find those capable of intelligent action.

This is an empirical hypothesis. We have defined a class of systems; we wish to ask whether that class accounts for a set of phenomena we find in the real world. Intelligent action is everywhere around us in the biological world, mostly in human behavior. It is a form of behavior we can recognize by its effects whether it is performed by humans or not. The hypothesis could indeed be false. Intelligent behavior is not so easy to produce that any system will exhibit it willy-nilly. Indeed, there are people whose analyses lead them to conclude either on philosophical or on scientific grounds that the hypothesis *is* false. Scientifically, one

can attack or defend it only by bringing forth empirical evidence about the natural world.

We now need to trace the development of this hypothesis and look at the evidence for it.

Development of the Symbol System Hypothesis

A physical symbol system is an instance of a universal machine. Thus the symbol system hypothesis implies that intelligence will be realized by a universal computer. However, the hypothesis goes far beyond the argument, often made on general grounds of physical determinism, that any computation that is realizable can be realized by a universal machine, provided that it is specified. For it asserts specifically that the intelligent machine is a symbol system, thus making a specific architectural assertion about the nature of intelligent systems. It is important to understand how this additional specificity arose.

Formal Logic. The roots of the hypothesis go back to the program of Frege and of Whitehead and Russell for formalizing logic: capturing the basic conceptual notions of mathematics in logic and putting the notions of proof and deduction on a secure footing. This effort culminated in mathematical logic—our familiar propositional, first-order, and higher-order logics. It developed a characteristic view, often referred to as the "symbol game." Logic, and by incorporation all of mathematics, was a game played with meaningless tokens according to certain purely syntactic rules. All meaning had been purged. One had a mechanical, though permissive (we would now say nondeterministic), system about which various things could be proved. Thus progress was first made by walking away from all that seemed relevant to meaning and human symbols. We could call this the stage of formal symbol manipulation.

This general attitude is well reflected in the development of information theory. It was pointed out time and again that Shannon had defined a system that was useful only for communication and selection, and which had nothing to do with meaning. Regrets were expressed that such a general name as "information theory" had been given to the field, and attempts were made to rechristen it as "the theory of selective information"—to no avail, of course.

Turing Machines and the Digital Computer. The development of the first digital computers and of automata theory, starting with Turing's own work in the '30s, can be treated together. They agree in their view of what is essential. Let us use Turing's own model, for it shows the features well.

A Turing machine consists of two memories: an unbounded tape and a finite state control. The tape holds data, i.e. the famous zeroes and ones. The machine has a very small set of proper operations—read, write, and scan operations—on the tape. The read operation is not a data operation, but provides conditional branching to a control state as a function of the data under the read head. As we all know, this model contains the essentials of all computers, in terms of what they can do, though other computers with different memories and operations might carry out the same computations with different requirements of space and time. In particular, the model of a Turing machine contains within it the notions both of what cannot be computed and of universal machines—computers that can do anything that can be done by any machine.

We should marvel that two of our deepest insights into information processing were achieved in the thirties, before modern computers came into being. It is a tribute to the genius of Alan Turing. It is also a tribute to the development of mathematical logic at the time, and testimony to the depth of computer science's obligation to it. Concurrently with Turing's work appeared the work of the logicians Emil Post and (independently) Alonzo Church. Starting from independent notions of logistic systems (Post productions and recursive functions, respectively) they arrived at analogous results on undecidability and universality—results that were soon shown to imply that all three systems were equivalent. Indeed, the convergence of all these attempts to define the most general class of information processing systems provides some of the force of our conviction that we have captured the essentials of information processing in these models.

In none of these systems is there, on the surface, a concept of the symbol as something that *designates*. The data are regarded as just strings of zeroes and ones—indeed that data be inert is essential to the reduction of computation to physical process. The finite state control system was always viewed as a small controller, and logical games were played to see how small a state system could be used without destroying the universality of the machine. No games, as far as we can tell, were ever played to add new states dynamically to the finite control—to think of the control memory as holding the bulk of the system's knowledge. What was accomplished at this stage was half the principle of interpretation—showing that a machine could be run from a description. Thus, this is the stage of automatic formal symbol manipulation.

The Stored Program Concept. With the development of the second generation of electronic machines in the mid-forties (after the Eniac) came the stored program concept. This was rightfully hailed as a milestone, both conceptually and practically. Programs now can be data, and can be operated on as data. This capability is, of course, already implicit in the model of Turing: the descriptions are on the very same tape as the data. Yet the idea was realized only when machines acquired enough memory to make it practicable to locate actual programs in some internal place. After all, the Eniac had only twenty registers.

The stored program concept embodies the second

half of the interpretation principle, the part that says that the system's own data can be interpreted. But it does not yet contain the notion of designation—of the physical relation that underlies meaning.

List Processing. The next step, taken in 1956, was list processing. The contents of the data structures were now symbols, in the sense of our physical symbol system: patterns that designated, that had referents. Lists held addresses which permitted access to other lists—thus the notion of list structures. That this was a new view was demonstrated to us many times in the early days of list processing when colleagues would ask where the data were—that is, which list finally held the collections of bits that were the content of the system. They found it strange that there were no such bits, there were only symbols that designated yet other symbol structures.

List processing is simultaneously three things in the development of computer science. (1) It is the creation of a genuine dynamic memory structure in a machine that had heretofore been perceived as having fixed structure. It added to our ensemble of operations those that built and modified structure in addition to those that replaced and changed content. (2) It was an early demonstration of the basic abstraction that a computer consists of a set of data types and a set of operations proper to these data types, so that a computational system should employ whatever data types are appropriate to the application, independent of the underlying machine. (3) List processing produced a model of designation, thus defining symbol manipulation in the sense in which we use this concept in computer science today.

As often occurs, the practice of the time already anticipated all the elements of list processing: addresses are obviously used to gain access, the drum machines used linked programs (so called one-plus-one addressing), and so on. But the conception of list processing as an abstraction created a new world in which designation and dynamic symbolic structure were the defining characteristics. The embedding of the early list processing systems in languages (the IPLs, LISP) is often decried as having been a barrier to the diffusion of list processing techniques throughout programming practice; but it was the vehicle that held the abstraction together.

LISP. One more step is worth noting: McCarthy's creation of LISP in 1959–60 [McCarthy, 1960]. It completed the act of abstraction, lifting list structures out of their embedding in concrete machines, creating a new formal system with S-expressions, which could be shown to be equivalent to the other universal schemes of computation.

Conclusion. That the concept of the designating symbol and symbol manipulation does not emerge until the mid-fifties does not mean that the earlier steps were either inessential or less important. The total concept is the join of computability, physical realizability (and by multiple technologies), universality, the symbolic representation of processes (i.e. interpretability), and, finally, symbolic structure and designation. Each of the steps provided an essential part of the whole.

The first step in this chain, authored by Turing, is theoretically motivated, but the others all have deep empirical roots. We have been led by the evolution of the computer itself. The stored program principle arose out of the experience with Eniac. List processing arose out of the attempt to construct intelligent programs. It took its cue from the emergence of random access memories, which provided a clear physical realization of a designating symbol in the address. LISP arose out of the evolving experience with list processing.

The Evidence

We come now to the evidence for the hypothesis that physical symbol systems are capable of intelligent action, and that general intelligent action calls for a physical symbol system. The hypothesis is an empirical generalization and not a theorem. We know of no way of demonstrating the connection between symbol systems and intelligence on purely logical grounds. Lacking such a demonstration, we must look at the facts. Our central aim, however, is not to review the evidence in detail, but to use the example before us to illustrate the proposition that computer science is a field of empirical inquiry. Hence, we will only indicate what kinds of evidence there is, and the general nature of the testing process.

The notion of physical symbol system had taken essentially its present form by the middle of the 1950's, and one can date from that time the growth of artificial intelligence as a coherent subfield of computer science. The twenty years of work since then has seen a continuous accumulation of empirical evidence of two main varieties. The first addresses itself to the *sufficiency* of physical symbol systems for producing intelligence, attempting to construct and test specific systems that have such a capability. The second kind of evidence addresses itself to the *necessity* of having a physical symbol system wherever intelligence is exhibited. It starts with Man, the intelligent system best known to us, and attempts to discover whether his cognitive activity can be explained as the working of a physical symbol system. There are other forms of evidence, which we will comment upon briefly later, but these two are the important ones. We will consider them in turn. The first is generally called artificial intelligence, the second, research in cognitive psychology.

Constructing Intelligent Systems. The basic paradigm for the initial testing of the germ theory of disease was: identify a disease; then look for the germ. An analogous paradigm has inspired much of the research in artificial intelligence: identify a task domain calling for intelligence; then construct a program for a digital computer

that can handle tasks in that domain. The easy and well-structured tasks were looked at first: puzzles and games, operations research problems of scheduling and allocating resources, simple induction tasks. Scores, if not hundreds, of programs of these kinds have by now been constructed, each capable of some measure of intelligent action in the appropriate domain.

Of course intelligence is not an all-or-none matter, and there has been steady progress toward higher levels of performance in specific domains, as well as toward widening the range of those domains. Early chess programs, for example, were deemed successful if they could play the game legally and with some indication of purpose; a little later, they reached the level of human beginners; within ten or fifteen years, they began to compete with serious amateurs. Progress has been slow (and the total programming effort invested small) but continuous, and the paradigm of construct-and-test proceeds in a regular cycle—the whole research activity mimicking at a macroscopic level the basic generate-and-test cycle of many of the AI programs.

There is a steadily widening area within which intelligent action is attainable. From the original tasks, research has extended to building systems that handle and understand natural language in a variety of ways, systems for interpreting visual scenes, systems for hand-eye coordination, systems that design, systems that write computer programs, systems for speech understanding—the list is, if not endless, at least very long. If there are limits beyond which the hypothesis will not carry us, they have not yet become apparent. Up to the present, the rate of progress has been governed mainly by the rather modest quantity of scientific resources that have been applied and the inevitable requirement of a substantial system-building effort for each new major undertaking.

Much more has been going on, of course, than simply a piling up of examples of intelligent systems adapted to specific task domains. It would be surprising and unappealing if it turned out that the AI programs performing these diverse tasks had nothing in common beyond their being instances of physical symbol systems. Hence, there has been great interest in searching for mechanisms possessed of generality, and for common components among programs performing a variety of tasks. This search carries the theory beyond the initial symbol system hypothesis to a more complete characterization of the particular kinds of symbol systems that are effective in artificial intelligence. In the second section of this paper, we will discuss one example of a hypothesis at this second level of specificity: the heuristic search hypothesis.

The search for generality spawned a series of programs designed to separate out general problem-solving mechanisms from the requirements of particular task domains. The General Problem Solver (GPS) was perhaps the first of these; while among its descendants are such contemporary systems as PLANNER and CONNIVER. The search for common components has led to generalized schemes of representation for goals and plans, methods for constructing discrimination nets, procedures for the control of tree search, pattern-matching mechanisms, and language-parsing systems. Experiments are at present under way to find convenient devices for representing sequences of time and tense, movement, causality and the like. More and more, it becomes possible to assemble large intelligent systems in a modular way from such basic components.

We can gain some perspective on what is going on by turning, again, to the analogy of the germ theory. If the first burst of research stimulated by that theory consisted largely in finding the germ to go with each disease, subsequent effort turned to learning what a germ was—to building on the basic qualitative law a new level of structure. In artificial intelligence, an initial burst of activity aimed at building intelligent programs for a wide variey of almost randomly selected tasks is giving way to more sharply targeted research aimed at understanding the common mechanisms of such systems.

The Modeling of Human Symbolic Behavior. The symbol system hypothesis implies that the symbolic behavior of man arises because he has the characteristics of a physical symbol system. Hence, the results of efforts to model human behavior with symbol systems become an important part of the evidence for the hypothesis, and research in artificial intelligence goes on in close collaboration with research in information processing psychology, as it is usually called.

The search for explanations of man's intelligent behavior in terms of symbol systems has had a large measure of success over the past twenty years; to the point where information processing theory is the leading contemporary point of view in cognitive psychology. Especially in the areas of problem solving, concept attainment, and long-term memory, symbol manipulation models now dominate the scene.

Research in information processing psychology involves two main kinds of empirical activity. The first is the conduct of observations and experiments on human behavior in tasks requiring intelligence. The second, very similar to the parallel activity in artificial intelligence, is the programming of symbol systems to model the observed human behavior. The psychological observations and experiments lead to the formulation of hypotheses about the symbolic processes the subjects are using, and these are an important source of the ideas that go into the construction of the programs. Thus, many of the ideas for the basic mechanisms of GPS were derived from careful analysis of the protocols that human subjects produced while thinking aloud during the performance of a problem-solving task.

The empirical character of computer science is nowhere more evident than in this alliance with psy-

chology. Not only are psychological experiments required to test the veridicality of the simulation models as explanations of the human behavior, but out of the experiments come new ideas for the design and construction of physical symbol systems.

Other Evidence. The principal body of evidence for the symbol system hypothesis that we have not considered is negative evidence: the absence of specific competing hypotheses as to how intelligent activity might be accomplished—whether by man or machine. Most attempts to build such hypotheses have taken place within the field of psychology. Here we have had a continuum of theories from the points of view usually labeled "behaviorism" to those usually labeled "Gestalt theory." Neither of these points of view stands as a real competitor to the symbol system hypothesis, and this for two reasons. First, neither behaviorism nor Gestalt theory has demonstrated, or even shown how to demonstrate, that the explanatory mechanisms it postulates are sufficient to account for intelligent behavior in complex tasks. Second, neither theory has been formulated with anything like the specificity of artificial programs. As a matter of fact, the alternative theories are sufficiently vague so that it is not terribly difficult to give them information processing interpretations, and thereby assimilate them to the symbol system hypothesis.

Conclusion

We have tried to use the example of the Physical Symbol System Hypothesis to illustrate concretely that computer science is a scientific enterprise in the usual meaning of that term: that it develops scientific hypotheses which it then seeks to verify by empirical inquiry. We had a second reason, however, for choosing this particular example to illustrate our point. The Physical Symbol System Hypothesis is itself a substantial scientific hypothesis of the kind that we earlier dubbed "laws of qualitative structure." It represents an important discovery of computer science, which if borne out by the empirical evidence, as in fact appears to be occurring, will have major continuing impact on the field.

We turn now to a second example, the role of search in intelligence. This topic, and the particular hypothesis about it that we shall examine, have also played a central role in computer science, in general, and artificial intelligence, in particular.

II. Heuristic Search

Knowing that physical symbol systems provide the matrix for intelligent action does not tell us how they accomplish this. Our second example of a law of qualitative structure in computer science addresses this latter question, asserting that symbol systems solve problems by using the processes of heuristic search. This generalization, like the previous one, rests on empirical evidence, and has not been derived formally from other premises. However, we shall see in a moment that it does have some logical connection with the symbol system hypothesis, and perhaps we can look forward to formalization of the connection at some time in the future. Until that time arrives, our story must again be one of empirical inquiry. We will describe what is known about heuristic search and review the empirical findings that show how it enables action to be intelligent. We begin by stating this law of qualitative structure, the Heuristic Search Hypothesis.

> *Heuristic Search Hypothesis.* The solutions to problems are represented as symbol structures. A physical symbol system exercises its intelligence in problem solving by search—that is, by generating and progressively modifying symbol structures until it produces a solution structure.

Physical symbol systems must use heuristic search to solve problems because such systems have limited processing resources; in a finite number of steps, and over a finite interval of time, they can execute only a finite number of processes. Of course that is not a very strong limitation, for all universal Turing machines suffer from it. We intend the limitation, however, in a stronger sense: we mean *practically* limited. We can conceive of systems that are not limited in a practical way, but are capable, for example, of searching in parallel the nodes of an exponentially expanding tree at a constant rate for each unit advance in depth. We will not be concerned here with such systems, but with systems whose computing resources are scarce relative to the complexity of the situations with which they are confronted. The restriction will not exclude any real symbol systems, in computer or man, in the context of real tasks. The fact of limited resources allows us, for most purposes, to view a symbol system as though it were a serial, one-process-at-a-time device. If it can accomplish only a small amount of processing in any short time interval, then we might as well regard it as doing things one at a time. Thus "limited resource symbol system" and "serial symbol system" are practically synonymous. The problem of allocating a scarce resource from moment to moment can usually be treated, if the moment is short enough, as a problem of scheduling a serial machine.

Problem Solving

Since ability to solve problems is generally taken as a prime indicator that a system has intelligence, it is natural that much of the history of artificial intelligence is taken up with attempts to build and understand problem-solving systems. Problem solving has been discussed by philosophers and psychologists for two millenia, in discourses dense with the sense of mystery. If you think there is nothing problematic or mysterious about a symbol system solving problems, then you are

a child of today, whose views have been formed since mid-century. Plato (and, by his account, Socrates) found difficulty understanding even how problems could be *entertained*, much less how they could be solved. Let me remind you of how he posed the conundrum in the *Meno:*

> Meno: And how will you inquire, Socrates, into that which you know not? What will you put forth as the subject of inquiry? And if you find what you want, how will you ever know that this is what you did not know?

To deal with this puzzle, Plato invented his famous theory of recollection: when you think you are discovering or learning something, you are really just recalling what you already knew in a previous existence. If you find this explanation preposterous, there is a much simpler one available today, based upon our understanding of symbol systems. An approximate statement of it is:

> To state a problem is to designate (1) a *test* for a class of symbol structures (solutions of the problem), and (2) a *generator* of symbol structures (potential solutions). To solve a problem is to generate a structure, using (2), that satisfies the test of (1).

We have a problem if we know what we want to do (the test), and if we don't know immediately how to do it (our generator does not immediately produce a symbol structure satisfying the test). A symbol system can state and solve problems (sometimes) because it can generate and test.

If that is all there is to problem solving, why not simply generate at once an expression that satisfies the test? This is, in fact, what we do when we wish and dream. "If wishes were horses, beggars might ride." But outside the world of dreams, it isn't possible. To know how we would test something, once constructed, does not mean that we know how to construct it—that we have any generator for doing so.

For example, it is well known what it means to "solve" the problem of playing winning chess. A simple test exists for noticing winning positions, the test for checkmate of the enemy King. In the world of dreams one simply generates a strategy that leads to checkmate for all counter strategies of the opponent. Alas, no generator that will do this is known to existing symbol systems (man or machine). Instead, good moves in chess are sought by generating various alternatives, and painstakingly evaluating them with the use of approximate, and often erroneous, measures that are supposed to indicate the likelihood that a particular line of play is on the route to a winning position. Move generators there are; winning move generators there are not.

Before there can be a move generator for a problem, there must be a problem space: a space of symbol structures in which problem situations, including the initial and goal situations, can be represented. Move generators are processes for modifying one situation in the problem space into another. The basic characteristics of physical symbol systems guarantee that they can represent problem spaces and that they possess move generators. How, in any concrete situation they synthesize a problem space and move generators appropriate to that situation is a question that is still very much on the frontier of artificial intelligence research.

The task that a symbol system is faced with, then, when it is presented with a problem and a problem space, is to use its limited processing resources to generate possible solutions, one after another, until it finds one that satisfies the problem-defining test. If the system had some control over the order in which potential solutions were generated, then it would be desirable to arrange this order of generation so that actual solutions would have a high likelihood of appearing early. A symbol system would exhibit intelligence to the extent that it succeeded in doing this. Intelligence for a system with limited processing resources consists in making wise choices of what to do next.

Search in Problem Solving

During the first decade or so of artificial intelligence research, the study of problem solving was almost synonymous with the study of search processes. From our characterization of problems and problem solving, it is easy to see why this was so. In fact, it might be asked whether it could be otherwise. But before we try to answer that question, we must explore further the nature of search processes as it revealed itself during that decade of activity.

Extracting Information from the Problem Space. Consider a set of symbol structures, some small subset of which are solutions to a given problem. Suppose, further, that the solutions are distributed randomly through the entire set. By this we mean that no information exists that would enable any search generator to perform better than a random search. Then no symbol system could exhibit more intelligence (or less intelligence) than any other in solving the problem, although one might experience better luck than another.

A condition, then, for the appearance of intelligence is that the distribution of solutions be not entirely random, that the space of symbol structures exhibit at least some degree of order and pattern. A second condition is that pattern in the space of symbol structures be more or less detectible. A third condition is that the generator of potential solutions be able to behave differentially, depending on what pattern it detected. There must be information in the problem space, and the symbol system must be capable of extracting and using it. Let us look first at a very simple example, where the intelligence is easy to come by.

Consider the problem of solving a simple algebraic equation:

$$AX + B = CX + D$$

The test defines a solution as any expression of the form, $X = E$, such that $AE + B = CE + D$. Now one could use as generator any process that would produce numbers which could then be tested by substituting in the latter equation. We would not call this an intelligent generator.

Alternatively, one could use generators that would make use of the fact that the original equation can be modified—by adding or subtracting equal quantities from both sides, or multiplying or dividing both sides by the same quantity—without changing its solutions. But, of course, we can obtain even more information to guide the generator by comparing the original expression with the form of the solution, and making precisely those changes in the equation that leave its solution unchanged, while at the same time, bringing it into the desired form. Such a generator could notice that there was an unwanted CX on the right-hand side of the original equation, subtract it from both sides and collect terms again. It could then notice that there was an unwanted B on the left-hand side and subtract that. Finally, it could get rid of the unwanted coefficient $(A - C)$ on the left-hand side by dividing.

Thus by this procedure, which now exhibits considerable intelligence, the generator produces successive symbol structures, each obtained by modifying the previous one; and the modifications are aimed at reducing the differences between the form of the input structure and the form of the test expression, while maintaining the other conditions for a solution.

This simple example already illustrates many of the main mechanisms that are used by symbol systems for intelligent problem solving. First, each successive expression is not generated independently, but is produced by modifying one produced previously. Second, the modifications are not haphazard, but depend upon two kinds of information. They depend on information that is constant over this whole class of algebra problems, and that is built into the structure of the generator itself: all modifications of expressions must leave the equation's solution unchanged. They also depend on information that changes at each step: detection of the differences in form that remain between the current expression and the desired expression. In effect, the generator incorporates some of the tests the solution must satisfy, so that expressions that don't meet these tests will never be generated. Using the first kind of information guarantees that only a tiny subset of all possible expressions is actually generated, but without losing the solution expression from this subset. Using the second kind of information arrives at the desired solution by a succession of approximations, employing a simple form of means-ends analysis to give direction to the search.

There is no mystery where the information that guided the search came from. We need not follow Plato in endowing the symbol system with a previous existence in which it already knew the solution. A moderately sophisticated generator-test system did the trick without invoking reincarnation.

Search Trees. The simple algebra problem may seem an unusual, even pathological, example of search. It is certainly not trial-and-error search, for though there were a few trials, there was no error. We are more accustomed to thinking of problem-solving search as generating lushly branching trees of partial solution possibilities which may grow to thousands, or even millions, of branches, before they yield a solution. Thus, if from each expression it produces, the generator creates B new branches, then the tree will grow as B^D, where D is its depth. The tree grown for the algebra problem had the peculiarity that its branchiness, B, equaled unity.

Programs that play chess typically grow broad search trees, amounting in some cases to a million branches or more. (Although this example will serve to illustrate our points about tree search, we should note that the purpose of search in chess is not to generate proposed solutions, but to evaluate (test) them.) One line of research into game-playing programs has been centrally concerned with improving the representation of the chess board, and the processes for making moves on it, so as to speed up search and make it possible to search larger trees. The rationale for this direction, of course, is that the deeper the dynamic search, the more accurate should be the evaluations at the end of it. On the other hand, there is good empirical evidence that the strongest human players, grandmasters, seldom explore trees of more than one hundred branches. This economy is achieved not so much by searching less deeply than do chess-playing programs, but by branching very sparsely and selectively at each node. This is only possible, without causing a deterioration of the evaluations, by having more of the selectivity built into the generator itself, so that it is able to select for generation just those branches that are very likely to yield important relevant information about the position.

The somewhat paradoxical-sounding conclusion to which this discussion leads is that search—successive generation of potentional solution structures—is a fundamental aspect of a symbol system's exercise of intelligence in problem solving but that amount of search is not a measure of the amount of intelligence being exhibited. What makes a problem a problem is not that a large amount of search is required for its solution, but that a large amount *would* be required if a requisite level of intelligence were not applied. When the symbolic system that is endeavoring to solve a problem knows enough about what to do, it simply proceeds directly towards its goal; but whenever its knowledge becomes inadequate, when it enters terra incognita, it

is faced with the threat of going through large amounts of search before it finds its way again.

The potential for the exponential explosion of the search tree that is present in every scheme for generating problem solutions warns us against depending on the brute force of computers—even the biggest and fastest computers—as a compensation for the ignorance and unselectivity of their generators. The hope is still periodically ignited in some human breasts that a computer can be found that is fast enough, and that can be programmed cleverly enough, to play good chess by brute-force search. There is nothing known in theory about the game of chess that rules out this possibility. Empirical studies on the management of search in sizable trees with only modest results make this a much less promising direction than it was when chess was first chosen as an appropriate task for artificial intelligence. We must regard this as one of the important empirical findings of research with chess programs.

The Forms of Intelligence. The task of intelligence, then, is to avert the ever-present threat of the exponential explosion of search. How can this be accomplished? The first route, already illustrated by the algebra example, and by chess programs that only generate "plausible" moves for further analysis, is to build selectivity into the generator: to generate only structures that show promise of being solutions or of being along the path toward solutions. The usual consequence of doing this is to decrease the rate of branching, not to prevent it entirely. Ultimate exponential explosion is not avoided—save in exceptionally highly structured situations like the algebra example—but only postponed. Hence, an intelligent system generally needs to supplement the selectivity of its solution generator with other information-using techniques to guide search.

Twenty years of experience with managing tree search in a variety of task environments has produced a small kit of general techniques which is part of the equipment of every researcher in artificial intelligence today. Since these techniques have been described in general works like that of Nilsson [1971], they can be summarized very briefly here.

In serial heuristic search, the basic question always is: what shall be done next? In tree search, that question, in turn, has two components: (1) from what node in the tree shall we search next, and (2) what direction shall we take from that node? Information helpful in answering the first question may be interpreted as measuring the relative distance of different nodes from the goal. Best-first search calls for searching next from the node that appears closest to the goal. Information helpful in answering the second question—in what direction to search—is often obtained, as in the algebra example, by detecting specific differences between the current nodal structure and the goal structure described by the test of a solution, and selecting actions that are relevant to reducing these particular kinds of differences. This is the technique known as means-ends analysis, which plays a central role in the structure of the General Problem Solver.

The importance of empirical studies as a source of general ideas in AI research can be demonstrated clearly by tracing the history, through large numbers of problem solving programs, of these two central ideas: best-first search and means-ends analysis. Rudiments of best-first search were already present, though unnamed, in the Logic Theorist in 1955. The General Problem Solver, embodying means-ends analysis, appeared about 1957—but combined it with modified depth-first search rather than best-first search. Chess programs were generally wedded, for reasons of economy of memory, to depth-first search, supplemented after about 1958 by the powerful alpha beta pruning procedure. Each of these techniques appears to have been reinvented a number of times, and it is hard to find general, task-independent theoretical discussions of problem solving in terms of these concepts until the middle or late 1960's. The amount of formal buttressing they have received from mathematical theory is still miniscule: some theorems about the reduction in search that can be secured from using the alpha-beta heuristic, a couple of theorems (reviewed by Nilsson [1971]) about shortest-path search, and some very recent theorems on best-first search with a probabilistic evaluation function.

"Weak" and "Strong" Methods. The techniques we have been discussing are dedicated to the control of exponential expansion rather than its prevention. For this reason, they have been properly called "weak methods"—methods to be used when the symbol system's knowledge or the amount of structure actually contained in the problem space are inadequate to permit search to be avoided entirely. It is instructive to contrast a highly structured situation, which can be formulated, say, as a linear programming problem, with the less structured situations of combinatorial problems like the traveling salesman problem or scheduling problems. ("Less structured" here refers to the insufficiency or nonexistence of relevant theory about the structure of the problem space.)

In solving linear programming problems, a substantial amount of computation may be required, but the search does not branch. Every step is a step along the way to a solution. In solving combinatorial problems or in proving theorems, tree search can seldom be avoided, and success depends on heuristic search methods of the sort we have been describing.

Not all streams of AI problem-solving research have followed the path we have been outlining. An example of a somewhat different point is provided by the work on theorem-proving systems. Here, ideas imported from mathematics and logic have had a strong influence on the direction of inquiry. For example, the use of heuristics was resisted when properties of com-

pleteness could not be proved (a bit ironic, since most interesting mathematical systems are known to be undecidable). Since completeness can seldom be proved for best-first search heuristics, or for many kinds of selective generators, the effect of this requirement was rather inhibiting. When theorem-proving programs were continually incapacitated by the combinatorial explosion of their search trees, thought began to be given to selective heuristics, which in many cases proved to be analogues of heuristics used in general problem-solving programs. The set-of-support heuristic, for example, is a form of working backwards, adapted to the resolution theorem proving environment.

A Summary of the Experience. We have now described the workings of our second law of qualitative structure, which asserts that physical symbol systems solve problems by means of heuristic search. Beyond that, we have examined some subsidiary characteristics of heuristic search, in particular the threat that it always faces of exponential explosion of the search tree, and some of the means it uses to avert that threat. Opinions differ as to how effective heuristic search has been as a problem solving mechanism—the opinions depending on what task domains are considered and what criterion of adequacy is adopted. Success can be guaranteed by setting aspiration levels low—or failure by setting them high. The evidence might be summed up about as follows. Few programs are solving problems at "expert" professional levels. Samuel's checker program and Feigenbaum and Lederberg's DENDRAL are perhaps the best-known exceptions, but one could point also to a number of heuristic search programs for such operations research problem domains as scheduling and integer programming. In a number of domains, programs perform at the level of competent amateurs: chess, some theorem-proving domains, many kinds of games and puzzles. Human levels have not yet been nearly reached by programs that have a complex perceptual "front end": visual scene recognizers, speech understanders, robots that have to maneuver in real space and time. Nevertheless, impressive progress has been made, and a large body of experience assembled about these difficult tasks.

We do not have deep theoretical explanations for the particular pattern of performance that has emerged. On empirical grounds, however, we might draw two conclusions. First, from what has been learned about human expert performance in tasks like chess, it is likely that any system capable of matching that performance will have to have access, in its memories, to very large stores of semantic information. Second, some part of the human superiority in tasks with a large perceptual component can be attributed to the special-purpose built-in parallel processing structure of the human eye and ear.

In any case, the quality of performance must necessarily depend on the characteristics both of the problem domains and of the symbol systems used to tackle them. For most real-life domains in which we are interested, the domain structure has not proved sufficiently simple to yield (so far) theorems about complexity, or to tell us, other than empirically, how large real-world problems are in relation to the abilities of our symbol systems to solve them. That situation may change, but until it does, we must rely upon empirical explorations, using the best problem solvers we know how to build, as a principal source of knowledge about the magnitude and characteristics of problem difficulty. Even in highly structured areas like linear programming, theory has been much more useful in strengthening the heuristics that underlie the most powerful solution algorithms than in providing a deep analysis of complexity.

Intelligence Without Much Search

Our analysis of intelligence equated it with ability to extract and use information about the structure of the problem space, so as to enable a problem solution to be generated as quickly and directly as possible. New directions for improving the problem-solving capabilities of symbol systems can be equated, then, with new ways of extracting and using information. At least three such ways can be identified.

Nonlocal Use of Information. First, it has been noted by several investigators that information gathered in the course of tree search is usually only used *locally,* to help make decisions at the specific node where the information was generated. Information about a chess position, obtained by dynamic analysis of a subtree of continuations, is usually used to evaluate just that position, not to evaluate other positions that may contain many of the same features. Hence, the same facts have to be rediscovered repeatedly at different nodes of the search tree. Simply to take the information out of the context in which it arose and use it generally does not solve the problem, for the information may be valid only in a limited range of contexts. In recent years, a few exploratory efforts have been made to transport information from its context of origin to other appropriate contexts. While it is still too early to evaluate the power of this idea, or even exactly how it is to be achieved, it shows considerable promise. An important line of investigation that Berliner [1975] has been pursuing is to use causal analysis to determine the range over which a particular piece of information is valid. Thus if a weakness in a chess position can be traced back to the move that made it, then the same weakness can be expected in other positions descendant from the same move.

The HEARSAY speech understanding system has taken another approach to making information globally available. That system seeks to recognize speech strings by pursuing a parallel search at a number of different

levels: phonemic, lexical, syntactic, and semantic. As each of these searches provides and evaluates hypotheses, it supplies the information it has gained to a common "blackboard" that can be read by all the sources. This shared information can be used, for example, to eliminate hypotheses, or even whole classes of hypotheses, that would otherwise have to be searched by one of the processes. Thus, increasing our ability to use tree-search information nonlocally offers promise for raising the intelligence of problem-solving systems.

Semantic Recognition Systems. A second active possibility for raising intelligence is to supply the symbol system with a rich body of semantic information about the task domain it is dealing with. For example, empirical research on the skill of chess masters shows that a major source of the master's skill is stored information that enables him to recognize a large number of specific features and patterns of features on a chess board, and information that uses this recognition to propose actions appropriate to the features recognized. This general idea has, of course, been incorporated in chess programs almost from the beginning. What is new is the realization of the number of such patterns and associated information that may have to be stored for master-level play: something of the order of 50,000.

The possibility of substituting recognition for search arises because a particular, and especially a rare, pattern can contain an enormous amount of information, provided that it is closely linked to the structure of the problem space. When that structure is "irregular," and not subject to simple mathematical description, then knowledge of a large number of relevant patterns may be the key to intelligent behavior. Whether this is so in any particular task domain is a question more easily settled by empirical investigation than by theory. Our experience with symbol systems richly endowed with semantic information and pattern-recognizing capabilities for accessing it is still extremely limited.

The discussion above refers specifically to semantic information associated with a recognition system. Of course, there is also a whole large area of AI research on semantic information processing and the organization of semantic memories that falls outside the scope of the topics we are discussing in this paper.

Selecting Appropriate Representations. A third line of inquiry is concerned with the possibility that search can be reduced or avoided by selecting an appropriate problem space. A standard example that illustrates this possibility dramatically is the mutilated checkerboard problem. A standard 64 square checkerboard can be covered exactly with 32 tiles, each a 1×2 rectangle covering exactly two squares. Suppose, now, that we cut off squares at two diagonally opposite corners of the checkerboard, leaving a total of 62 squares. Can this mutilated board be covered exactly with 31 tiles? With (literally) heavenly patience, the impossibility of achieving such a covering can be demonstrated by trying all possible arrangements. The alternative, for those with less patience, and more intelligence, is to observe that the two diagonally opposite corners of a checkerboard are of the same color. Hence, the mutilated checkerboard has two less squares of one color than of the other. But each tile covers one square of one color and one square of the other, and any set of tiles must cover the same number of squares of each color. Hence, there is no solution. How can a symbol system discover this simple inductive argument as an alternative to a hopeless attempt to solve the problem by search among all possible coverings? We would award a system that found the solution high marks for intelligence.

Perhaps, however, in posing this problem we are not escaping from search processes. We have simply displaced the search from a space of possible problem solutions to a space of possible representations. In any event, the whole process of moving from one representation to another, and of discovering and evaluating representations, is largely unexplored territory in the domain of problem-solving research. The laws of qualitative structure governing representations remain to be discovered. The search for them is almost sure to receive considerable attention in the coming decade.

Conclusion

That is our account of symbol systems and intelligence. It has been a long road from Plato's *Meno* to the present, but it is perhaps encouraging that most of the progress along that road has been made since the turn of the twentieth century, and a large fraction of it since the midpoint of the century. Thought was still wholly intangible and ineffable until modern formal logic interpreted it as the manipulation of formal tokens. And it seemed still to inhabit mainly the heaven of Platonic ideals, or the equally obscure spaces of the human mind, until computers taught us how symbols could be processed by machines. A.M. Turing, whom we memorialize this morning, made his great contributions at the mid-century crossroads of these developments that led from modern logic to the computer.

Physical Symbol Systems. The study of logic and computers has revealed to us that intelligence resides in physical symbol systems. This is computer sciences's most basic law of qualitative structure.

Symbol systems are collections of patterns and processes, the latter being capable of producing, destroying and modifying the former. The most important properties of patterns is that they can designate objects, processes, or other patterns, and that, when they designate processes, they can be interpreted. Interpretation means carrying out the designated process. The two most significant classes of symbol systems with which we are acquainted are human beings and computers.

Our present understanding of symbol systems grew, as indicated earlier, through a sequence of stages. Formal logic familiarized us with symbols, treated syntactically, as the raw material of thought, and with the idea of manipulating them according to carefully defined formal processes. The Turing machine made the syntactic processing of symbols truly machine-like, and affirmed the potential universality of strictly defined symbol systems. The stored-program concept for computers reaffirmed the interpretability of symbols, already implicit in the Turing machine. List processing brought to the forefront the denotational capacities of symbols, and defined symbol processing in ways that allowed independence from the fixed structure of the underlying physical machine. By 1956 all of these concepts were available, together with hardware for implementing them. The study of the intelligence of symbol systems, the subject of artificial intelligence, could begin.

Heuristic Search. A second law of qualitative structure for AI is that symbol systems solve problems by generating potential solutions and testing them, that is, by searching. Solutions are usually sought by creating symbolic expressions and modifying them sequentially until they satisfy the conditions for a solution. Hence symbol systems solve problems by searching. Since they have finite resources, the search cannot be carried out all at once, but must be sequential. It leaves behind it either a single path from starting point to goal or, if correction and backup are necessary, a whole tree of such paths.

Symbol systems cannot appear intelligent when they are surrounded by pure chaos. They exercise intelligence by extracting information from a problem domain and using that information to guide their search, avoiding wrong turns and circuitous bypaths. The problem domain must contain information, that is, some degree of order and structure, for the method to work. The paradox of the *Meno* is solved by the observation that information may be remembered, but new information may also be extracted from the domain that the symbols designate. In both cases, the ultimate source of the information is the task domain.

The Empirical Base. Artificial intelligence research is concerned with how symbol systems must be organized in order to behave intelligently. Twenty years of work in the area has accumulated a considerable body of knowledge, enough to fill several books (it already has), and most of it in the form of rather concrete experience about the behavior of specific classes of symbol systems in specific task domains. Out of this experience, however, there have also emerged some generalizations, cutting across task domains and systems, about the general characteristics of intelligence and its methods of implementation.

We have tried to state some of these generalizations this morning. They are mostly qualitative rather than mathematical. They have more the flavor of geology or evolutionary biology than the flavor of theoretical physics. They are sufficiently strong to enable us today to design and build moderately intelligent systems for a considerable range of task domains, as well as to gain a rather deep understanding of how human intelligence works in many situations.

What Next? In our account today, we have mentioned open questions as well as settled ones; there are many of both. We see no abatement of the excitement of exploration that has surrounded this field over the past quarter century. Two resource limits will determine the rate of progress over the next such period. One is the amount of computing power that will be available. The second, and probably the more important, is the number of talented young computer scientists who will be attracted to this area of research as the most challenging they can tackle.

A.M. Turing concluded his famous paper on "Computing Machinery and Intelligence" with the words:

> "We can only see a short distance ahead, but we can see plenty there that needs to be done."

Many of the things Turing saw in 1950 that needed to be done have been done, but the agenda is as full as ever. Perhaps we read too much into his simple statement above, but we like to think that in it Turing recognized the fundamental truth that all computer scientists instinctively know. For all physical symbol systems, condemned as we are to serial search of the problem environment, the critical question is always: What to do next?

References

Berliner, H. [1975]. Chess as problem solving: the development of a tactics analyzer. Ph.D. Th., Computer Sci. Dep., Carnegie-Mellon U. (unpublished).

McCarthy, J. [1960]. Recursive functions of symbolic expressions and their computation by machine. *Comm. ACM 3*, 4 (April 1960), 184–195.

McCulloch, W.S. [1961]. What is a number, that a man may know it, and a man, that he may know a number. *General Semantics Bulletin* Nos. 26 and 27 (1961), 7–18.

Nilsson, N.J. [1971]. *Problem Solving Methods in Artificial Intelligence*. McGraw-Hill, New York.

Turing, A.M. [1950]. Computing machinery and intelligence. *Mind 59* (Oct. 1950), 433–460.