Artificial Intelligence: Where Has It Been, and Where is it Going?

Herbert A. Simon

Abstract—**Artificial intelligence was born in close connection with management science, grew apart from it, and is now forming new links with it, as well as with the other disciplines that have come together in cognitive science. AI specialists account for only a fraction of the expert systems that have been built, and this will continue to be true in the future—the applications of AI are too wide-ranging to be left to AI specialists alone.**

The directions for near-future development of AI can be described in terms of four dichotomies: the use of reasoning versus the use of knowledge; the roles of parallel and of serial systems; systems that perform and systems that learn to perform; and programming languages derived from the search metaphor versus languages derived from the logical reasoning metaphor. There is no question of "either-or." Although I believe that there are reasons for emphasizing knowledge systems (production systems) that are serial, capable of expert performance, and designed in terms of the search metaphor, the other pathways are also important and should not be ignored. In particular, we need empirical work—the construction and empirical testing of the performance of large systems—to explore all of these branching pathways.

Index Terms—**Artificial intelligence, expert systems, heuristic search, list processing, logic languages, machine learning, serial versus parallel programs.**

I THOUGHT it would be appropriate this morning in introducing a conference on Applications of AI to Management & Economics to address my remarks not only to that general area but to spend a good deal of my time categorizing what AI is all about and also what the relationship is between AI and some of the other tools that have been provided to managers and to the use of economic researchers and policy makers; tools which also in many cases rely on the computers but nevertheless are not usually regarded as part of AI.

Before I start in on that, I must say that my confidence in AI is very much raised this morning by the knowledge that my meals on Singapore Airlines were planned in that way. The system was so effective and so flexible that when we had a schedule change and certain portions of the flight took place at different times from when we expected them to take place, the meals also somehow or other magically changed. We had appropriate meals for all times of the day so I can recommend that particular AI system if it is available for export to other applications.

Manuscript received October 1, 1989; revised October 1, 1990. This work was supported by the Defense Advanced Research Project Agency, Department of Defense, ARPA Order 3597, monitored by the Air Force Avionics Laboratory under Contract F33615-81-K-1539. This paper was delivered as the Keynote Speech at the 2nd International Workshop on Artificial Intelligence in Economics and Management at the Institute of Systems Science, National University of Singapore, January 9–13, 1989.

The author is with the Department of Psychology, Carnegie Mellon University, Pittsburgh, PA 15213.

IEEE Log Number 9144300.

On the History of AI

We all tend to use a common definition of artificial intelligence when we are asked about it. What is artificial intelligence? We call a program for a computer "artificially intelligent" if it does something which, when done by a human being, will be thought to require human intelligence. We define it in terms of the tasks that are done.

But from a professional and sociological standpoint, that, of course, is not quite accurate, because there are many things that computers do which when done by a human being would be regarded as requiring intelligence (and maybe even rather high intelligence), but which nevertheless have never been considered to be artificial intelligence. From a disciplinary standpoint as well as the standpoint of sociology, we are concerned not with all kinds of intelligence on machines but with a subset of them. For example, computers for a very long time have been inverting matrices and solving partial differential equations, but we usually don't regard that as artificial intelligence. It used to be thought to require certain amounts of human intelligence. Graduate students in some fields were supposed to be able to invert matrices and I can remember in the age before computers (there was really an age before computers; only a few of us in this room can remember that), one had to master the Gauss–Seidel method to solve simultaneous equations and invert matrices. We thought we were using our intelligence, but nowadays perhaps 90% of the effort of computers is devoted to numerical analysis and we don't think of that as artificial intelligence. I am not making an imperialistic statement here, I am not alleging that we should claim it, but just pointing out to you that our field is really defined in a rather peculiar way.

Then of course there is operations research, whose origin predates artificial intelligence by perhaps a decade. Here again the computer is used to perform sophisticated tasks, solving problems in linear programming for example. If we didn't know any better, we might suspect that linear programming was an artificial intelligence technique. After all what is linear programming except a form of heuristic search? You start out in a big convex space and you start moving through that space from one facet of the convex surface to another until you get to the optimal point. Of course, this is not heuristic because there is a theorem that proves you always get there sooner or later, and maybe that's why it isn't regarded as artificial intelligence.

1041–4347/91/0600–0128$01.00 © 1991 IEEE

Also, there is a whole field of what might well be called "do-it-yourself" artificial intelligence, because from the very beginning, users of computers who got access to them for other reasons—for doing numerical analysis—have often conceived ways they could use the computers' intelligence, properly programmed, to do their work. As far back as 1960, the Westinghouse Electric Co. in the United States—I am sure there are a lot of others, but I just happen to know this example—was doing routine motor design, transformer design, and generator design by computers automatically, and you will find a couple of early publications by a man named Godwin around 1960 describing those computer design programs. I don't think Mr. Godwin at that time had ever heard of artificial intelligence. He just happened to have a computer in his design office and the people there thought: well, we have this crude method for taking a customer's order and looking at the specifications of a motor and going through the empirical tables for determining what the windings of that motor should be and so on and so forth . . . why don't we just put that all into the computer and do it automatically . . . and then they were doing it. I mention this because that puts us who are in the field of artificial intelligence in a rather peculiar professional position. A lot of others are practicing our profession without a license.

The numerical analysts, the operation researchers, and the increasing number of do-it-yourself artificial intelligence people indicate, in a way, the importance of artificial intelligence. It is too important to be left to the artificial intelligence specialists just as computer science in general is far too important to be left to computer scientists. AI is a tool which like mathematics or, for that matter, like language, spreads through the whole society. Everybody has to use this tool and many people are going to be creative in its use, not just the people who fancy themselves to be professionals. Not all the writing in the world is done by professors of English or of Chinese or of any natural language. So we are in that peculiar position where a particular subset of the intelligent things that computers do are singled out and called artificial intelligence and the boundaries are really quite arbitrary. As a matter of fact, it often turns out that after we have solved a number of problems by artificial intelligence techniques and if we then develop a theory about it so that everything gets to be very neat and tidy, then somehow it is no longer artificial intelligence—it is mathematics or something else. AI is sort of a residual category. I think the field is always going to have that character and I don't think there is anything to be alarmed about or anything to be disappointed about.

In the early days of artificial intelligence, it had a very close relation to numerical analysis and particularly to operations research. If you look back at the early volumes of the *Operations Research Journal,* or the *Journal of the Institute of Management Science* (TIMS), you will find some of the classical papers of artificial intelligence.

Tonge's assembly line balancing system was first developed in a context where nobody made any distinction between management science and artificial intelligence, and the same is true of the first portfolio selection program of Geoffrey Clarkson. Both of these were done around 1960. In the classical Feigenbaum and Feldman book *Computers and Thought* you will find chapters on those two programs.

After a few years, artificial intelligence and operations research grew apart. I don't know if anybody has done a serious job of trying to understand the history. I viewed it happening at my own university—Carnegie-Mellon University—and it happened in the following way: when computers first came into the university, through a historical accident, they came in through the basement of the business school. That was where we housed the first computer, and we were soon doing artificial intelligence research on it around 1956–1958. So some of the first applications of artificial intelligence were to business management, and the same faculty that was doing that were also using operation research tools.

After a few years, Computer Science became a separate entity. Drawing on other departments—the Business School, Psychology Department, Electrical Engineering Department, Mathematics Department—it became very interdisciplinary, then spun off as a separate field taking artificial intelligence along with it. There then began to be a split between the two, and if you examine the *Operations Research Journal* and the *TIMS Journal* you will find a steady decrease in the articles published there on heuristic programming, heuristic search techniques, and so forth. There were quite a few such articles in the early days. That split is very unfortunate, because if we look at modern management and its need for sharp tools to assist managerial judgment, we see that those tools have to be drawn both from the classical methods of operations research like linear programming, integer programming, and queueing theory, and increasingly from the methods of artificial intelligence.

In the past 3 or 4 years—maybe a little more than that, but since the time that artificial intelligence has begun to get a lot of public attention—I have sensed a drawing together again of these fields. One can give various interpretations of that convergence. A cynical interpretation would be that OR consultants began to notice that they were getting competition from consultants who were building expert systems and decided that it was time to pay a little attention to artificial intelligence techniques and to add those to their kit of tools in advising management and giving expert services to management. So increasingly again today, and I think it is a very good development, we are beginning to see artificial intelligence skills being brought together with traditional management science and operations research skills. So much for the relation of artificial intelligence to management science and operations research.

Let me say just a word about AI's relation to cognitive science. Cognitive science of course is also a new word. We have had for a somewhat longer time cognitive psychology. Cognitive psychology is simply the study of human thinking and it has been, although not always under that label, a traditional part of the science of psychology. What cognitive science does today is to bring together a variety of people who are interested in the general study of intelligence—that includes, of course, not only the cognitive psychologists, it includes the artificial intelligence community; it includes linguists; it includes a great many philosophers (epistemology

and other areas of philosophy are really taking on new life in the light of what they are learning about artificial intelligence); and it includes various sorts of other people such as anthropologists, and increasingly will include people in the field of language, particularly people interested in literary criticism and in what is now called rhetoric—the new rhetoric.

So cognitive science is a meeting place in which all of those can come together and we have a series of bridges here: Cognitive science is bridging from artificial intelligence over into the human sciences, while artificial intelligence is bridging over to the natural sciences through computer science and to mathematics and to computer engineering, and we get a great span of activity all of which is extremely relevant to artificial intelligence. We can't all be experts in all of those things, but I think we can all be aware of all of them and can be alert to the possibilities of getting useful ideas from those domains.

Let me just mention a few examples of the flow of ideas between cognitive science and artificial intelligence. I am going to make several predictions this morning. I have to be careful about predictions for the future. A few of you in the room here know that Allen Newell and I in 1957 in the *Operations Research Journal* made some famous predictions. The predictions were all right but the times we assigned to them weren't all right. So now when I make predictions, I am a little more careful about putting particular numbers on them. I might be off by a factor of 3—even meteorologists don't do better than that.

My prediction in this case is that one of the areas in artificial intelligence where we will progress very rapidly in the next few years, is in the capability of computers to use natural languages. In advancing there, we have had close ties with the linguistics community which has helped us with the syntactic aspects of this relation, because that is what they themselves have mostly focused on. We have gained help in the language area from logicians to some extent; those who think that human reasoning has some relationship to logic. I will have something to say about that a little later—as to whether there is any close relation between formal logic and human reasoning. But increasingly now I think we will be looking to other people who have been examining natural language. I mentioned the rhetoricians.

Rhetoric is a name revived from Greek times for the study of the nature of argumentation in natural language prose (i.e., how do people write prose or speak in an effort to persuade other people). What is the nature of that flow of language? There are a lot of interesting things to be learned from the literature of rhetoric, and for that matter the literature of literacy criticism, and this is going to have an impact on artificial intelligence. Some of you who are interested in that area had better start reading some of the classics of the new rhetoric.

Another example of that cross-tie is the technique of protocol analysis—a technique of getting people to perform tasks and think aloud while they are performing those tasks, and of using that information in the building of expert systems. Protocol analysis came out of psychology as a technique used for the study of problem solving. A good deal of underlying theoretical work had to be done to understand what you can find out from protocols and, equally important, what you *can't* find out from protocols. The field of artificial intelligence has now absorbed those techniques, and I think improved on them as well, and they have become a very important part of the set of skills that are required in building expert systems.

I might remind you of list processing languages including the language Lisp and others, which borrow the idea of storing things in computer memory in associative structures. You don't use the natural addressing system of the computer but you link things by "next" relations into lists and list structures. That whole idea was originally suggested by the well-known fact that the human mind is associative (i.e., that the thought of one thing leads to another) and by the attempt to capture that property in a computer language so that you could write programs where you couldn't anticipate in advance what memory structure you were going to build. If you are inverting a matrix you know exactly what memory cells you are going to need; if you are building a search tree, you have no idea what is going to happen and we needed this kind of associative structure and the idea came initially from cognitive psychology.

The idea of neural nets, some of which are more neural than others, is again an idea that came out of psychology research. One of the first networks within this scope that was computerized, was Quillian's network, an attempt to explain how people search an associative memory. Other network researchers followed on from that. Other approaches like the Perceptrons were very much influenced by Hebb's work in experimental psychology which proposed some things called cell assemblies as a basic structure for human memory. The idea of object-based programming I think is also due to this series of ideas, because one of the things the Lisp programming did, was to allow you to build up various kinds of schemas. Everybody has a pet name for that, I call them schemas; Marvin Minsky prefers to call them frames; at Yale they are scripts, I don't know what they are called in Singapore, but I call them schemas because that is the name that holds historical precedence. That idea again was realized first in list processing languages.

The idea of heuristics and of heuristic search comes directly out of the psychological literature on problem solving, particularly the European psychologists, the Gestaltists, people like Duncker who had a fairly clear picture of means end analysis long before computers arrived. He had as clear a picture as you can get without having the precise languages of computers to say exactly what you mean, but Duncker was certainly a forerunner of these developments.

In expert systems themselves, the idea that to create an expert you have to store away an awful lot of information in the form of condition–action pairs or productions; and that if you add a modest inference engine you probably can get some expert behavior out of the system—as indeed we have found that we can in many many applications. That idea came in part out of research on such human activities as chess playing, where we found that expert chess players were doing just that, and that raised confidence that we could do the same thing with computers. I don't mean that all of these ideas arose in psychology and then got transferred over into computer

science. There was a very strong flow of ideas back and forth between the two.

Moving on from cognitive science, I mentioned earlier the relation of artificial intelligence to logic in the area of language. Of course, that is not the only area where that influence has been present and many of the pioneers of artificial intelligence, starting with John McCarthy, really got their inspiration from logic. Those of you who are Lisp programmers know that Lisp is sort of a hybrid which involves mating a particular form of logic known as the lambda calculus with the idea of list structures. As a matter of fact in the early days if you took a Lisp course you always took it at your own risk because, depending on which instructor you got you, either got a course in the lambda calculus or a course in list structures and you seldom got a course in both. That division of Lisp is still apparent in the textbooks. If you look at your Lisp textbooks, you will see chapters that came out of Church's lambda calculus and chapters that came out of list processing. But logic exercises an even stronger influence today on what is usually called logic programming.

The idea of logic programming is that all the inference that takes place in artificial intelligence (and maybe also in human intelligence) consists in proving things, and that what humans do in their reasoning is to move from premises to conclusions using the laws of logic. A language like Prolog tries to capture that idea. The Prolog language tries to get you to state everything you know in a bunch of propositions, a bunch of declarative statements. It then says "we give you this handy little inference engine which will perform resolution on Horn clauses and prove everything you want to know." There is also the idea that has been promoted a good deal by John McCarthy, that since classical logics are truth preserving (a statement once proved is immortally true), and since the world consists of fleeting facts which become true and then cease to be true, we therefore need logics that are able to handle this so called nonmonotonicity, this changing of truth values from true to false and back again. There has been some development of nonmonotonic logic. All of these are very important influences on artificial intelligence; ideas that not everybody in artificial intelligence uses, but which have influenced important pieces of AI research.

I have indicated a few of the relations of artificial intelligence to other fields to illustrate the complexities of the field we are in. Artificial intelligence is a wonderful vantage point if you are intellectually curious; a wonderful vantage point from which to view the world, a view from the Middle Kingdom. You can see over a very wide horizon; you can move into all the human sciences; you can move into the natural sciences. It is a nice place to be and I think we all enjoy it.

Issues for the Future

Now having said those things about the past, and since all of you know quite well how AI stands at the moment, I would like to devote the rest of my remarks to some notions about the future. I have titled them here "Issues for the Future." This is going to be a rather opinionated review of these issues, because I am going to make some evaluative statements about where I think things are going, and where I think things ought to go. These may not be the views you hold, but I am going to leave plenty of time this morning for discussion.

I have put the issues under four headings and you will see they overlap, but this will give us an idea of some of the things we have to decide when we are choosing which AI techniques to develop, what kind of research is needed, what kinds of tools are available for application. These are all issues that already have been raised; they are not new issues that I have thought up at this moment. They are all in the literature and there already has been some discussion of them.

I will put them in terms of antinomies (this versus that) but you have to interpret the "versus" very cautiously because I don't think we are talking exclusively about one or the other of each of these alternatives, but how they should work together.

First is, reasoning versus knowledge. This is Ed Feigenbaum's theme song. The early decades of AI, he says, emphasized generality; how you reason about anything or how you think about anything. In a way GPS, the General Problem Solver, symbolizes that era of artificial intelligence research. And then, says Ed, came the Knowledge Era when you realized that for expert behavior in any domain, you need lots of knowledge.

But we still have to decide in building our systems what role is going to be given to thinking and what role is to be given to knowledge. We know that a very bright person can get along with sketchier knowledge than a person who is less bright because the bright person can fill in the gaps of knowledge by thinking about how it must be; by deducing things one from another. Don't push that too far. I notice that when people get out of the fields in which they have very good knowledge, their inferences get shakier and shakier. In general, we need to have some combination of inferencing capability and knowledge, and every expert system makes provision for that.

Second is the very large issue of serial systems versus parallel systems, and you know who the popular favorite is at the present time. AI started out, possibly because of the hardware available, with serial architectures, that is, one thing after another. Today there is a very high interest in moving from those serial systems to parallel systems. That is issue number 2. I am just naming them now—I will say something about them later.

The third is performance versus learning. Almost all of the applications of AI up until a couple of years ago were in the form of performance programs—programs that can do something, but you have to program them to do it. And then the idea comes along (actually the idea goes a long way back, but aside from Samuel's checker program, there were very few successful implementations of it): why program these dull things, why don't we just let the computers learn, put the burden of learning on their shoulders—that's what we do with human beings so why not do it with computers? I want to raise that question of when we prefer to build performance programs and when we prefer to build learning programs.

The last set of issues, I don't have it in the form of "versus"; it has to do with programming languages: what should our programming languages look like?

Let me take these issues up briefly one by one. First reasoning versus knowledge. I made the point already that in any system that is to do anything interesting nowadays, we need a combination of a lot of knowledge and some capability of drawing inferences from that knowledge. This is shown by the characteristic anatomy of expert systems, which have a bunch of productions that embody the knowledge (if you see spots on the patient, then you suspect the following diseases, and so forth) and some capability of making inferences.

If you are going to make inferences, then that gets us immediately into the question of the forms of reasoning. How do we make these inferences? Maybe I should hesitate a little bit about calling them "forms of reasoning" because that prejudices the issue. The word "reasoning" has always been connected with logic. Let's call it for the moment "forms of thinking." What should be the capabilities of an expert system or any other kind of AI system? What should its capability be for thinking, and how do we implement it?

One very popular approach, particularly in the Eastern Hemisphere and Europe, is Logic Programming. Logic Programming, in the form of Prolog or other languages, draws upon the metaphors of logic: that thinking is sort of like doing deduction in logic and therefore, for thinking on a computer, we ought to give the computer the capability of doing deduction in logic. Now of course a programming language like Prolog is not committed to behaving like a logician. As a matter of fact, very earlier on, pure Prolog was blemished by adding certain features like the "cut" feature which enables you to write "real" programs in Prolog. (That's a combatative remark.) Prolog as it is usually used in practice is not a pure logic language, nor used just to deduce things from Horn clauses, but actually has the capabilities of a perfectly general programming language.

As a matter of fact you can say this about almost all programming languages. They are all equivalent to Turing machines by the time you put the bells and whistles on them. And then what does it matter what programming language is used? Well, it matters because the philosophy underlying your programming language influences the style of your programming. You can program in Prolog almost identically with the way you program in Lisp—but you don't. You program in a different way because the Prolog language was designed to lead you into a different way of thinking about programming. It is a different metaphor of programming. It is a metaphor that says a program is something that proves things, that proves theorems; whereas Lisp gives you a metaphor that a program is something for moving symbolic structures around in memory and modifying those symbolic structures.

The two languages suggest very different metaphors and therefore lead to writing very different programs. If you follow the philosophy of Prolog, you try to put everything in the form of declarative statements, and let the built-in inference engine do all the inferencing for you. You don't try to control the course of the computation. You do this at your own peril, but that is what you do if you write pure Prolog.

If you use other languages, like production system languages, for example (OPS-5), then you go to the other extreme—you put everything or almost everything in procedural form. All your knowledge of the world turns out to be a set of "if–thens." That leads to a very different program architecture, although you clearly can do the same things in both languages. You might ask for a cue from human performance as to which is better. After all most anthropologists believe human beings have been thinking for about a million years, so it may be worthwhile to find out how we do it. This does not compel us to do it the same way on computers but at least it suggests that there might be some advantages or disadvantages to do it one way or another.

John R. Anderson, my colleague in the Psychology Department, has made the following interesting observation on the basis of his research on students doing geometry problems. What John finds, and what other people have found in other contexts, is that very frequently, people first learn things in declarative form. I don't think it universally true but it often happens. They first learn things in declarative form. For example, they learn theorems of geometry—e.g., if two triangles have two angles and the included side equal then the two triangles are equal. You learn that declaratively and then you prove things using the theorem interpretively, that is, you have an interpreter which can take those declarative sentences and modify them using rules of inference to get other things. But what Anderson also finds is that as people become more skilled in something like geometry, their declarative knowledge is transformed into procedural knowledge. It moves from being a set of propositions into being a set of productions. If that is the case (and it is frequently the case) then we might ask whether perhaps there is some advantage of having things in the form of productions rather than having them in declarative form. Then we might take the further scandalous step of asking whether these same advantages might hold for computers as they do for human beings. Of course, if we do that, we might end up thinking that there are some reasons for using production systems in many cases rather than using logic languages.

But what might be the reason why having things in procedural form is more advantageous than having them in declarative form? Everybody knows that if you compile a program, you gain an order of magnitude in speed (rule of thumb). So what happens according to this picture that John Anderson is drawing, is that we get our knowledge in interpretable form on which we apply general rules of inference (our GPS or Prolog type of system) and then we transform it into compiled form which runs a lot faster, so an expert geometrician does not have to go through this translation.

There is another way of seeing this. Perhaps when you were in high school and took a physics course, you had to do pulley problems, in which there are some arrangements of pulleys—a weight hanging from one of them and a weight hanging on a rope on the other end and the whole thing was supposed to be in balance. These were the usual frictionless pulleys that physicists have in the laboratories—you never find them anywhere else. And you had to compute the forces that were balancing it—you will be given the size of one weight and you have to compute the balancing weight. You can go back to Newton's three laws of motion, particularly the third law,

and run through that computation if you know how to apply it to pulleys and to ropes. On ropes it is very tricky because each molecule of the rope is pulling on the next molecule and if you really compute that all the way through, you will be a long time getting to the end of the rope.

What does the physicist do? The physicist has acquired some productions, so that when he looks at a diagram in which there is a rope over a pulley, the physicist already has a compiled production which says "if I know the force over on one side, I can assign the same force to the other side otherwise the thing wouldn't be in balance." On any pulley that is in balance, the numerical force on one side of the rope has to be the same as the other. As a matter of fact, for a rope running over a bunch of pulleys, the force has to be everywhere the same or the rope would move. Now you don't have to go through that reasoning each time. All you need is a production that, when it looks at a pulley or at a diagram of pulleys stored in the computer memory and notices a rope that has a known force on it, it immediately assigns that force to every other piece of that rope no matter where it is in the diagram. You can solve pulley problems very fast by having that production and two others—three productions will do the trick, with no reasoning, no syllogisms, nothing of that sort.

So there is good reason to believe that human experts do not solve problems by taking a bunch of propositions and then applying *modus ponens* to them or even unit resolution. Human beings are filled with specialized inference rules in the form of productions, and those rules are tuned to each of the tasks we do. They are not universal laws of logic but they are task dependent conditionally true rules that are only true in a particular domain. I wonder whether we can give up that insight in our artificial intelligence programs? If that is the case of human thinking, then we gain exactly those same advantages by proceeding this way in computer languages and that gives me a very deep concern about where we are going with logic programming. I will be glad to discuss with you whether there is any reason to think that is the way to go.

Let me give you one other related example. The Stanford Research Institute a few years ago built a system called "STRIPS"—most of you know about it. It was a system that was supposed to do the planning of the behavior of Shaky, which was the robot that Stanford Research Institute had built to wander through some rooms and bring your newspaper back to you or whatever it was supposed to do. Shaky was supposedly controlled by STRIPS which was going to plan its behavior and tell it what to do next. When that project started out, the idea was that STRIPS would be a logic programming scheme. It was supposed to prove a theorem, the steps of the theorem being essentially the plan for Shaky; and the theorem to be proved being the goal that you were trying to reach. This is the usual way in which you set up problem solving as theorem proving. And then it turned out that it was simply an infeasible scheme because it got into a computational problem that maybe could have been done with a super computer but were really horrendous. So when Fikes got involved in this project, he proposed to divide the reasoning in STRIPS into two parts.

First of all Shaky was moving from place to place, so was changing its situation. If you tried to describe its situation, then clearly your logic was going to be nonmonotonic because the situation was changing all the time. It is no longer true that it is in the living room, now it is in the dining room, and now that isn't true anymore because it is now in the kitchen. So you first of all have to have a nonmonotonic logic. Fikes asked how other people do this, including economists and physicists. What they do is simply to put a time subscript on the variables and to define something called the state or state of affairs. They describe how things are at one state of affairs, and then they apply some processes which in STRIPS are called "move operators" to move to another state of affairs. We won't go through the technicalities here.

You can proceed in this way without going through the whole process of reasoning about what all the things are that are still true in the new state of affairs, and what things aren't true in the new state of affairs. The move operator does this automatically for you and so the project of controlling Shaky changed from a project in logic programming to a project in modeling a situation and applying move operators to change that situation: in other words, to use the classical model of heuristic search. This is just one example, and perhaps the lessons that are applied there don't apply to a large number of other tasks. But at least before we acquire an unmitigated enthusiasm for logic programming, we ought to look at some of the lessons from the past and see whether that is really the way to go.

I obviously do have an opinion on that. I think that the future is the future of search, of modeling (sometimes called mental modeling), of building descriptions of situations and move operators that change those descriptions and hence change the situation. This is the way to solve the problem of nonmonotonic logic and not by trying to build special modal logics of one kind or another—something that nobody has really brought off today.

Let me move to "serial versus parallel" and give you a few more of my opinions. I needn't point out to you the enthusiasm for parallel systems today, and I think that this direction needs to be very vigorously researched. I don't really need to say that, because given the enthusiasm, it *will* be very vigorously researched. There are a zillion proposals today for various kinds of parallel systems—some of them having the characteristic of networks and some of them quite different. Among networks we can find a great number of varieties, some of them bearing a resemblance to neuron-like things and others bearing no resemblance to neuron-like things, or a little resemblance. So I am talking about a very heterogeneous class of systems here.

Well again, let's start with some questions about human architecture. Here's this creature that has been thinking for a million or two years but doing something else for longer than that. Its mammalian ancestors have been seeing and hearing and moving for about 400 million years. The so-called higher mental functions—the things we use as professionals, the professors use and ministers of state use, people like that—are a million or two years old.

It is very obvious, first of all, that biologically and anatomically, the brain is a parallel device. All the neurons, 10 billion or more, seem to be blinking all the time. So clearly we have a device with a lot of parallelism in it.

On the other hand, if we start by not looking at the brain but looking at human behavior that requires attention (let's say driving a car) then it turns out that human beings are very serial one-at-a-time systems, or perhaps, occasionally two at a time. You can drive a car and carry on a conversation at the same time if the traffic isn't too dense, but be sure you have your priorities right. Most of us who do that are aware that we are really engaged not in parallel computation but in time sharing. Now I have exaggerated that a little, because there is automation of human skills, some skills get to be practised automatically and don't seem to require conscious attention after a while, and so there may be other parallel activities going on. But still I think what is most impressive, if you take a general view of this, is how serial we are, how we have to move from one goal to another goal and attend to one of them at a time. That's just a very central fact of human life. That could be due to two reasons.

One, it might be due to the fact that we are badly designed. Evolution has only had a million years for thinking, but 400 million years to design our eyes. A million years for evolution isn't very long, and the cerebrum is probably just a jerry built device. It just gets by, whereas the human eyes and ears and the connection with the motor system, those are highly tuned by 400 million years of evolution and they have been designed to use parallelism all the time. There is no doubt at all that the eyes are parallel devices that can handle the quanta of energy that are hitting the retina in many places at a time. They are also a very special purpose parallel device—so special purpose that we have to have a different mechanism for our ears. So we do have parallel devices and they do seem to be specially designed to deal with particular functions; and as the information comes in through these highly parallel devices, it is apparently recoded and soon it's controlled by the serial machine and is no longer being operated on in parallel fashion.

As I said, that might be because we were badly designed. The other possibility is that there is a reason for it. What could the reason be? The reason just could be this fact: If you think of most goal oriented activities, and particularly if you think of planning for goal oriented activities, then in the process of planning there are an awful lot of precedence relations. There are an awful lot of cases where it is not worth thinking about B until you have thought about A. Therefore, there is no use in carrying out the computation in parallel.

Current ideas of parallel computation for chess playing might be and already are becoming a very interesting area of research on what parallelism and serialism are all about. There are a number of very good chess playing programs which use multiple processors. One of the most powerful in the world today (High Tech) uses 64 processors, each taking care of moves originating on one square of the board. That is a very powerful computing system and I doubt whether it gets more than a 5 times increase in speed over a one processor machine.

I attended a meeting about a year ago where I was listening to people describe quite a number of parallel systems with 30 processors or something of the sort, and the typical result was that you get a speed factor of 5 out of 30 processors, maybe a factor of 6 out of 60 processors and so on. Clearly it depends on the task, and in chess playing for example, some of the interesting experimentation now going on is whether you can reallocate tasks from one machine to another dynamically and therefore recapture a much higher efficiency in the system. That gets you into a very nice job shop scheduling problem, which of course has never been solved by OR techniques and which people are now trying to solve with artificial intelligence techniques.

So I am not, by any means, saying: "Let's forget about all this parallel stuff." What I am saying is that the prospect of building general purpose parallel machines and getting any kind of a high factor of efficiency in their use seems to me very dubious, whereas the possibility of building parallel machines adapted to particular classes of tasks (machines that know something about the priority relations in those tasks and know something about how to reallocate capacity dynamically), those possibilities are large indeed.

There is another hint we might get for parallel systems from natural systems, which has gotten a little attention (but to my mind surprisingly little). Nature uses lots of parallel systems. I've got zillions of cells in my body and all of them are doing something almost all of the time–metabolizing, I guess that's what cells do. But one of the things we find about the parallel systems in nature is that they are almost all hierarchical in architecture. What I mean by hierarchical is that there are tiny subsystems that interact with each other like mad. Let's start with the very beginning with quarks. The quarks are interacting to form particles like protons and neutrons, and then the protons and neutrons interact with each other to form nucleons and atoms and a few electrons. And these interact with each other to form molecules and we can keep going up. Now by the time you get to atoms, you are no longer concerned with quarks. In other words, the atoms work as though they are just protons and neutrons, and you don't need to know the details of the quarks. And for doing chemistry you don't need to know the details of the neutrons most of the time and so on.

Each of these levels is nearly sealed off from the level below and only aggregated information from the lower levels gets transmitted to the higher levels of the system. As a matter of fact, a formal theory has been built to discuss the mathematics of this, the mathematics of nearly decomposable systems. If you want to see some computer science applications of the theory, applications to the analysis of operating systems, you can look at the book by Courtois called *Decomposable Systems* published by McGraw Hill about 15 years ago.

The theory tells you why it is that in hierarchical systems, all you need to know about the smallest components, if you are concerned with the higher components, are simply aggregated aspects of their behavior. And it also tells you that systems have to be designed so that most of the interactions, particularly the high frequency interactions take place within the smallest clusters, the next level of frequency within the immediate clusters, the slower frequencies within the higher clusters and so forth—you can have as many layers as you like.

For those of you who are physicists, there is an interesting relation here which nobody has fully laid out yet, with the so called renormalization theory in physics. My point about this is that parallelism is indeed an exciting area and I don't think we are talking about parallelism in general, I think we are talking about many different kinds of possible parallel architectures and I think we are talking about adapting those architectures to special classes of tasks.

The original supposition about supercomputers was that they were going to be built for artificial intelligence. I don't know of any artificial intelligence being done on supercomputers today. Maybe there is some but I don't know of any. A large amount of artificial intelligence is being done on PC's or workstations. What is being done on supercomputers is numerical computation of equation systems that have certain very strong regularities that allow you to make use of the parallel operations without being stifled by the precedence relations. So these general purpose parallel machines (I shouldn't just be speaking of supercomputers which are only a little bit parallel—but also other architectures now coming into the stream with large amounts of parallelism) largely find themselves programmable for certain classes of tasks where the precedence relations are compatible with the architecture of the machine. This is a lesson for machine architects, but also a lesson for those of us who are doing software and who are attempting to use these machines.

Finally let me just say a few words about performance versus learning. It has been pointed out that human beings are not programmed. We learn, so why not computers? Or maybe the question we ought to ask is, "why human beings?" Just suppose we could open up a person's head and put in a program for algebra or Latin or whatever it is you want to study in the schools, close the box again and then that person would be able to do algebra or Latin. That would be a great thing. Just think of what we could save in the State Budget if we did it that way. Let's get all those kids out of school at age 8 and get them to work to do something useful in the society.

But instead we go through this tortuous process from age 6 up to . . . well I won't say anything about graduate students . . . in which we give the system certain inputs (often in declarative form as though they were doing logic programming) and have it gradually convert them into productions and store them so that it can recognize situations where the information is relevant and then do something about it. Nobody in his right mind would do it that way if it could be programmed directly. So why do we want learning for computers? Well, there might be other reasons why we would want learning.

I should point out that no matter how hard the programming is, once we got the system programmed, then it is available essentially to all computers of the right size in the world. So there is no use having a hundred computers learning. At most we can just send one to school and the others can be working. Then when the one learns what it has to learn, we simply copy the program into the others. One should not conclude from this that I think we should give up research on learning for computers. I can think of at least two reasons why we should not give it up.

First of all, it is very hard to teach or program something you don't know and so a lot of what we call learning on computers is really discovery. We clearly want to give computers the capacity to discover new things because that is an important skill that humans also have to have and there is no way we can open up a box and put in the raw materials and get the discovery out without having a program that does it. We need programs that can behave in a creative manner and make scientific discoveries. That is one reason to pursue learning research, though it might be less confusing to call it discovery research or something of that sort to indicate the objective.

There is another reason for pursuing learning research although it is still a bit obscure. More and more, our computer systems have software which has quite a long life. I have had the experience of operating on a particular computer over a period of 15 years. By that time it was an antiquated computer, but as far as I am concerned I would rather work on an antiquated computer than learn a new operating system, particularly if it is UNIX. So I kept using this computer until they put it to sleep and I had to move to another. Now in that period of 15 years, goodness knows what sort of things had gotten into the memory of that computer. There were hardly any people around, by the time it was dead, who had put the stuff in. It was put in by generations of graduate students who had moved on to better things. So nobody knew what was in that computer and debugging it and improving programs and adding things to it without getting horrendous interactions became harder and harder. I don't have to tell you this. You all live in such situations every day.

And so the idea comes up that it would be very nice to have a computing system where you can add things without knowing what is in there already, without knowing the structure of the existing program. Well that's what learning is all about. What do teachers do? We have to know a little bit of what the students know in general terms—it is not very much use teaching a student calculus if he hasn't had algebra. So we have to know something about the starting condition. We certainly don't have to know how this information is organized in the human mind. If I were doing research on learning for computers as distinct from discovery, I guess I would put my emphasis on saying "what could this contribute to systems that live through accretion, systems that live by constant addition of new information to them, where you really don't want to have to know the details of the organization of the information inside." And I think that is a legitimate reason for doing research on learning.

I think that deals with most of the issues I want to cover. I have already had my say about logic programming as against list processing or as against production systems. I also talked about parallel computers which also applies to languages for executing programs, so I don't think I have anything else to say about languages. Let me draw some conclusions that will temper a little bit some of the strong opinions I have been expressing.

The first conclusion that I would draw is that we really must let a hundred flowers bloom, maybe a thousand flowers. The whole character of this field, now barely 30 or 35 years old, is that it is very much in its infancy and we simply have

to pursue that whole range of exciting lines that are opened to us. It doesn't matter who is right or wrong, we will find that out soon enough and it usually turns out that everybody is half right and half wrong—some with bigger halves than others, but it doesn't matter. The whole nature of scientific research is that if you know the answers, why do the research; if you don't know the answers, you have to pursue alternative possibilities, and if there are bright people who think there are possibilities that look promising, you better give those bright people the means to pursue them. So I think our field is in a very healthy state as long as we have a good competition going between parallelism and serialism, between logic programming and procedural programming, between all those "versuses" that I have listed here.

A second point I would make, which I haven't discussed much in my talk, is that we have, I think, a very good style of research in artificial intelligence. It is a style that has been criticized a good deal and there certainly are some improvements that can be made. I will mention one or two.

This style is a very empirical style. We have been behaving as though computer science or at least artificial intelligence, is like biology or agronomy or something like that. You plant a lot of flowers and see what grows. Our main tool of research has been to build systems and see how they operate. Some people who come to us out of mathematics, or who don't come to us but sit on the other side of the fence and heckle us, say "how can you do that, what's the good of a science unless it has a theory?" and we all look back at Newton's three laws of motion.

We are never going to have three laws of motion in computer science. Instead take as a model something like molecular biology, full of facts with a little bit of theory to hang those facts together. There are some great pieces of theory in biology like the structure of the DNA chain and so forth. But the bread and butter of a molecular biologist is to get into a laboratory and find facts. Now computing systems may or may not be quite as complicated as living organisms but they are pretty complicated, and the principal way in which we are going to learn about them is to go into a laboratory and find facts. We do that by building systems and testing them. That is not an argument against theory (I have even been caught doing a little theorizing once or twice). But the theory is about optimal search. It says that theory can only be realistically based on adequate experimentation, so let's just continue to build systems and test them.

We can improve our method of building them. We can standardize our test beds a good deal more than we have, so that we get information we can compare from one situation to another. We can do a better job of describing systems particularly software systems, so that other people can understand them. We can do a better job of making the software available and properly documented so that it becomes a part of the scientific environment and not just a private possession.

I will really worry about the future of Artificial Intelligence on the day that it decides to become a theoretical science—it might go the way of Economics, a fate worse than death. Well, enough said. When people ask "where's the theory," you can point them to the pieces of theory that are around—there are quite a number of them and I even have mentioned a couple of them this morning. But be relaxed about it. The theory will come along as we get a better understanding of the phenomena, and it will be fun doing theory too, but not at the expense of continuing to explore by building systems to widen the range of intelligent things that computers can do.

Herbert A. Simon received the Ph.D. degree from the University of Chicago, Chicago, IL, in 1943.

He is Professor of Computer Science and Psychology at Carnegie Mellon University, Pittsburgh, PA, where he has taught since 1949.

Dr. Simon received the Alfred E. Nobel Award in Economics in 1978, and the ACM Turing Award (with Allen Newell) in 1975. He is the author (with Newell) of *Human Problem Solving,* and (with Langley, Bradshaw, and Zytkow) of *Scientific Discovery: Computational Explorations of the Creative Processes.*