FANTASY, COMPUTATION, AND REALITY[1]

Nils J. Nilsson
Computer Science Department
Stanford University
Stanford, CA 94305

December 14, 1985

I. Introduction

The title of this essay hints that I do not intend it to be about computers in any expected sense. I'm not going to say anything about home computers, large "main frames," or how to set up a database system. I'm not going to talk so much about *what* you can do with computers as I am about *how* computers do what they do and why that's an important *intellectual* subject. My task is similar to that of a nuclear physicist wanting to describe his subject. Nuclear physics isn't only about hydrogen bombs or nuclear powerplants—it's an exciting subject in its own right aside from its engineering consequences. What I really want to discuss is the science that makes computers possible—that is, *computer science* or *computation*.

Computer science is a twentieth-century subject. It is not at all well understood yet by either the general public or by intellectuals. It promises (or threatens, depending on your point of view) to change man's view of himself even more profoundly than did the Copernican revolution or the theory of evolution. It brings previous philosophical discussions about the nature of the mind suddenly into sharper focus. It helps us also create a clearer picture of the relationship between the mind and the rest of the universe. Physics, chemistry and biology (all physical sciences) reigned supreme during the last few centuries. They produced our current understanding of the important fundamental components and processes of the physical world—the understanding on which our twentieth-century technological society is built. But I believe that the two most important scientific enterprises for the rest of this century and into the next are computer science and developmental genetics (the study of how genetic instructions operate). These two subjects—more related than one might think—will show us how we can (and how nature already does) aggregate the primitive building blocks of physics into unimaginably wonderful complexes such as human beings and futuristic robots.

In explaining computer science, I'm going to claim that it's subject matter is essentially *fantasy*, and that computation serves as a kind of bridge between fantasy and reality. In fact, computer science helps us understand that in a certain sense, fantasy is as close as we can get to reality. But before trying to make sense of these cryptic comments, I must first dispose of some popular fantasies about computers. By saying a few things about what computer science *isn't*, we'll be better prepared to understand what it *is*. I'm going to ask you to forget (at least for now) everything you might have known about computers while I take you on a journey describing them in a way you might never have heard before.

II. Some Popular Misconceptions About Computers

It's surprising to computer scientists to discover that so many people think that computers necessarily deal with *numbers*. Computers really process *symbols*. Those symbols can just as easily stand for letters, real people, geometric

[1] This paper is the written version of the 1985 Jacob Bronowski Memorial Lecture given at New College, University of Toronto on November 6, 1985. Copyright © Nils J. Nilsson, 1985.

shapes, or what have you. It’s not really any easier to have them stand for numbers—it’s just that having computers process numerals was such an important early and common use that many people think that’s *all* they can do. So, to those who presume that they *must* deal with a computer in the language of numbers, let me say “that’s not so at all!” A computer is a symbol cruncher—not only a number cruncher.

Many people also think that computers necessarily have something to do with *electricity*. Present day computers are electrical, but computers have been built that don’t use electricity. Some have used mechanical levers and gears. Others have used fluids like air or water. Computers have been proposed that would use laser light or complex chemicals like DNA molecules. What makes a computer a computer doesn’t depend on what it is made of. They don’t have to be electrical! Of course, computers do need to be embodied in *something*! The “ethereal” symbols processed by computers need *some* bridge to reality. It just happens that currently electronics is the medium by which the fantasy of computation gets connected to the reality of the actual world. But any limits peculiar to electricity (that might not apply to other physical phenomena) are not fundamental limits to computation.

A few leading intellectuals, such as the philosopher John Searle, think that human-like mental activity can occur only in certain *kinds* of materials, like the protein structures of nerve cells. That sort of idea goes entirely against the grain of a view widely held by artificial intelligence researchers called the *symbol system hypothesis*. That hypothesis affirms first that it is entirely immaterial how computation is implemented. The only important thing is the fact that symbols are manipulated by rule-like processes. Further, the hypothesis claims that *all* intelligent behavior can ultimately be explained by symbol-processing systems—none of it depends either on non-symbolic processes or on the special *physical* properties of the material in which the symbol manipulation takes place.

Most embodiments of computation have been in devices that have two states: *on* and *off*. Examples are electrical switches such as relays and transistors. For these devices, then, the all important bridge between symbol structures and their realization in the world involves translating our alphabet of symbols down into two basic ones represented by the two states of the physical device. But if we back away a bit from this bridge, into the territory where computer scientists do most of their thinking, we have an unlimited supply of different symbols. Since arbitrary collections of symbols can be translated into binary or two-symbol form, the *binaryness* of the bridge has no influence whatsoever on what computer scientists can do or think about. Some people imagine that because computers are limited to the impoverished language of two symbols—1 and 0, that computers have correspondingly limited expressiveness. Not so! Binaryness is not itself a limitation—it just happens to be one of the languages used to connect computer science to physics. It has been proven to be a universal language—one whose only limitations are due to the very nature of symbols themselves. Symbols have to be *discrete*. This means essentially that we have to be able to isolate and count them even if the count could go arbitrarily high. There are some very interesting mathematical theorems about the limitations of computers that stem from the discreteness of their symbols. But most artificial intelligence researchers think that these same theorems would apply to any physical symbol processors—even to *humans*!

Other misconceptions arise when computers are compared to the human brain. One often hears statements like “computers will never be able to do what the brain does, because computers use *this* and the brain uses *that*.” The *this* and *that* contrasts might be *binary* versus *non-binary* symbols, *serial* versus *parallel* processing, *silicon* versus *protein*, or any of a number of other alternatives. Well, we don’t really know yet what the brain uses, but we have already stated that the use of binary numbers doesn’t limit computers. It *is* true that many present computers go through their operations serially—one step after another and that much processing in the brain appears to occur in parallel. But the fundamental concepts of computer science are independent of this *implementational detail*, and we can be assured that if parallelism turns out to be an important engineering consideration, it will not greatly fluster computer scientists.

One also hears people say that computers can't make mistakes, and therefore they can't ever be as creative, or as unpredictable, or as irrational as humans are. Nonsense! It's all too easy to get computers to make all kinds of mistakes, but most mistakes (even those of humans) don't lead to anything worthwhile. We are remembering only our very few lucky mistakes and forgetting our many unfortunate blunders when we think that the secret of success lies in error!

One thing people have said about computers that *is* true is that computers can do only what they are told to do by their programmers. People shouldn't assume, however, that this statement implies that computers can't do anything that their programmers hadn't foreseen at the time of programming. Programmers can give computers exceedingly complex instructions—so complex that not even the programmers can predict what these instructions will cause the computer to do. Programmers can invent rules for producing novelty and can instruct computers with these rules. At a level higher, they can invent rules for inventing rules for producing novelty, and so on.

There are many other myths about computation, and we don't have space to go into all of them here. Be suspicious though, of statements like "computers can't do such and such because ...." The only one of these kinds of statements that need concern us here is "computers can't do such and such because we haven't yet figured out how to get them to do it."

III. Fundamentals of Computer Science

Nuclear physicists have as their subject matter the atomic nuclei and the processes they undergo. Civil engineers have as their subject matter the properties of materials and how they can be arranged into useful structures such as bridges and buildings. What is the *stuff* of computer science? It is not, as I have already mentioned, *silicon* or *electricity*. Computation deals with *structures* much as engineering does, but these structures are the abstract symbolic structures of mathematics and logic. They are products of the imagination, just as is the stuff of mathematics. The subject matter of computer science is *symbol structures*, their properties and what can be done with them and to them. Just as electrical engineering uses a knowledge of physics to build its electronic marvels, computer science uses the ideas of certain branches of mathematics.

Let's talk for a bit about symbol structures and how they are used by computer scientists. I'm not going to get very mathematical in this essay, so it's going to be a challenge to explain precisely what symbol structures are. They are like many things we already know about. They are like the phone list tacked to the kitchen wall. Or like the family tree of English Monarchs. Or like the chart of mileages between principal cities on the back of a road atlas. Or like the list of employees working for a major corporation. They are like recipes and musical scores. Although they aren't as solid as most things engineers deal with, they certainly are subjects ordinary people think about, talk about, and use all the time. Computer science invents and studies the abstract mathematical structures used to represent these lists, trees, and charts.

Computer scientists use their symbol structures for two different kinds of purposes. Most simply, we use them just to store information—such as data about employees, their salaries, and other records. But we also use them to store instructions. Take a musical score, for example. As a symbolic *structure*, it is used to pass down from one generation to the next the genius of Beethoven. But it can also be *played*. It can be *interpreted* as a set of instructions about how someone should behave with a musical instrument to produce the beautiful sounds that the score represents. The score itself is, in a way, a fantasy, an abstraction, just an imaginary structure of symbols. To be heard it must connect to reality through its rendition on physical instruments.

Recipes are structures of symbols similar to musical scores. A recipe can abstractly represent an apple pie, but it can also be used to instruct a chef about how to make one.

It's important to understand that it is completely irrelevant what the score or recipe is written on. We think of it as the *very same* score regardless of its being printed on parchment, on clay, on paper, on magnetic tape, or as holes in a player piano roll. Much the same can be said about the symbol structures of computer science. An important point about computer science, to reverse a famous Torontan's aphorism, is that "the medium is *not* the message." Because of their abstract nature, we often refer to collections of our symbol structures as *software*.

Symbols can instruct a wider class of beings than just us people. If represented in sufficiently detailed and precise form, they can also be interpreted by simple mechanical devices. A representation of Scott Joplin's music on a player piano roll can be converted by a player piano to the music itself. One of the antecedents of computers is the Jacquard Loom. The cards used to control the loom hold the symbol structures representing the pattern to be woven.

The player piano and the Jacquard loom are perfect examples of one aspect of computer science, namely the transformation of abstract, imaginative, symbolic instructions into concrete actions in the real world. I call this kind of software *stage one* software. It is used to store data and patterns, and it can guide machinery that transforms those patterns into sound, tapestry, and sculpted steel. Nascent computer science stayed at stage one for many years. The IBM corporation—formed from techniques invented by Herman Hollerith before the turn of the century to tabulate census data—used essentially only stage one software until the 1950s.

Computer science exploded out of its first stage with the realization that if software could be used as a set of instructions to act on the physical world, it could also be used as instructions to change symbol structures themselves! Let's call such symbol-changing software *stage two* software. The drama of this idea is particularly apparent when we realize that a list of instructions can be used to change the very symbol structure representing the instructions themselves! This possibility derives from John von Neumann's fundamental contribution to computer science, namely the proposal that the instructions for symbol manipulation be stored in the same memory as are stored the symbols to be manipulated.

Our intuition about using symbolic instructions to change a structure of symbols can be aided by looking at a simple but fascinating example that illustrates the complex changes that can follow from some very simple instructions. We look at mathematician John Conway's game called *Life*.[2] Imagine a rectangular grid of cells like that shown in Figure 1a—only larger. There, some of the cells are black and the rest are white. We can imagine this grid being represented by a structure of binary symbols. Now I am going to propose some instructions for changing this structure into a new one. In the new structure, a cell is black if in the old structure the corresponding cell was black and had two or three black immediate neighbors or if the corresponding cell was white and had exactly three black immediate neighbors. Otherwise, a cell in the new structure is white. (Instructions for computations are full of *ifs* and *otherwises*.) Suppose we apply these rules to the structure of Figure 1a. We get Figure 1b. Successive generations of these rule applications give us Figures 1c through 1e. Subsequent applications progressively change the structure. It is interesting to note that, viewing these structures as frames in a movie, it appears that a black projectile is moving off toward the southeast at a rate of one diagonal cell every four generations.

[2] See Martin Gardner, "Mathematical Games," *Scientific American*, October, 1970; or Martin Gardner, *Wheels, Life and Other Mathematical Amusements*, W. H. Freeman and Company, New York, 1983.

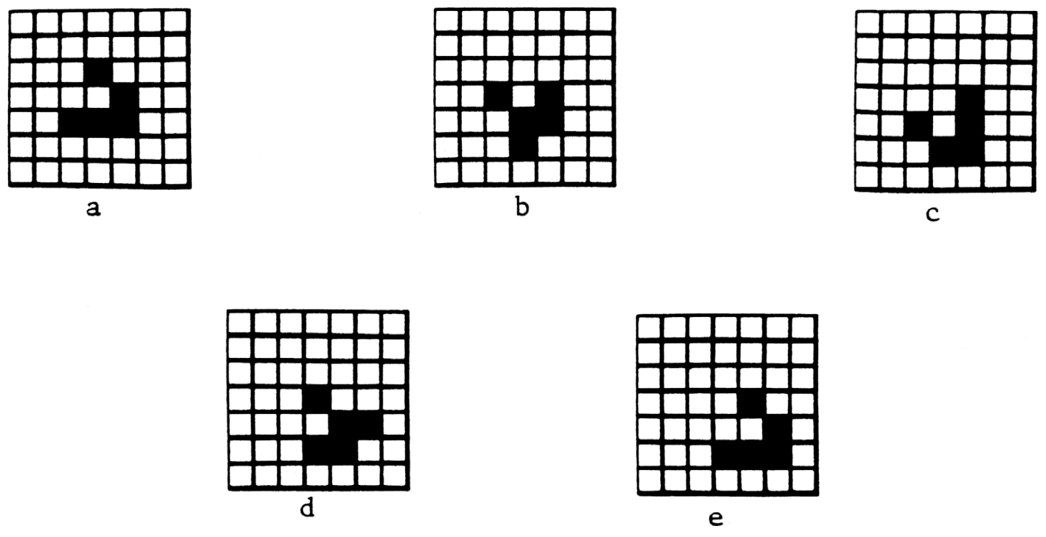

Figure 1. Stages in the Game of *Life*

This game is a simple example of how we use sets of instructions, called *programs*, to change symbol structures. In fact, *computation* (in its broadest sense) is mainly about the changes that can be made to symbol structures. There is a rich variety of primitive instructions used by conventional computers. Some of them, called *fetch* or *read* instructions, bring components of symbol structures into the *central processor* for subsequent manipulation. Others evoke primitive symbol-manipulating operations, much like those used in the *Life* game, that change symbol structures within the central processor. *Print* or *store* instructions transfer symbol structures out of the central processor into *memory* where they await further processing. Just as the instructions for a Jacquard loom are used to create complex patterns of cloth, computer programs work on symbol structures producing any conceivable modification to them—depending on the purposes of the programmer.

From the primitive operations used to change symbol structures, we can build up hierarchies of arbitrarily complex symbol-changing processes. Particularly useful assemblages of primitive instructions can be packaged in *subroutines* which can be used in more complex operations. Similarly, particularly useful structures of *atomic* symbols form molecules that in turn can be used as higher level building blocks. Much like a corporation's organization chart, the fundamental entities of computer science can be aggregated into higher level boxes that themselves are components of still higher level boxes. As we shall see, this layering process is an extremely important organizing principle for computer science. With it, we can design and build programs that alphabetize a list of our friends, or that insert a new name in the list, or that compute the total mileage traveled on our trip last summer. Building on only a small set of primitive operations, we can write programs that perform any conceivable computation. In fact, the *symbol system hypothesis* says, in effect, that *all* the aspects of intelligent behavior and thought can be achieved by computational symbol processing engines of this kind.

There is no need to limit our symbol structures to those that represent real things in the real world. They can represent anything we care to imagine from unicorns to perpetual motion machines. What might it have been like if our own family tree included a French duke who met and married an English poet. Let's put their offspring in our family tree giving us some imaginary distant cousins. So, besides the symbol structures themselves being abstract and nonphysical, they can be about unreal situations. We can give our imagination free play and create structures that represent whatever we want them to. And then, we can do anything imaginable to those structures. The very property that makes the subject matter of computer science so hard to make tangible is the property that makes it so useful!

Those familiar with home computers or personal computers used in business may have used *spread-sheet* programs in which one can try out various financial *scenarios*. None of them need represent situations that are real. Computation makes it easy for us to ask "what-if" questions about imaginary worlds. The answers to these questions help us decide what actions to take in the real world—guiding us toward realizeable fantasies and away from nightmares come true. Engineers can experiment with designs—simulating the real world by computations that predict how new structures and products would fare.

Chess-playing computers select moves by a systematic exploration of imaginary situations. We can illustrate the methods they use by looking at the simpler game of naughts and crosses (tic-tac-toe). In Figure 2 we show a symbol structure representing a situation in the game. It's a 3 x 3 array whose cells contain one of three symbols standing for *empty*, filled with a *cross*, or filled with a *naught*. Our computation explores 5 *alternative futures*—those representing what it would be like if a cross were placed in each essentially different spot. The naughts-and-crosses computation works its way down a tree of this sort estimating the consequences of possible moves in order to select one. These kinds of programs exploit a lesson already learned by evolution: it's best to explore first a dream world of computation before risking disaster in real life. Computation helps us make fantasies come true by telling us what we have to do to affect the world in desirable ways.

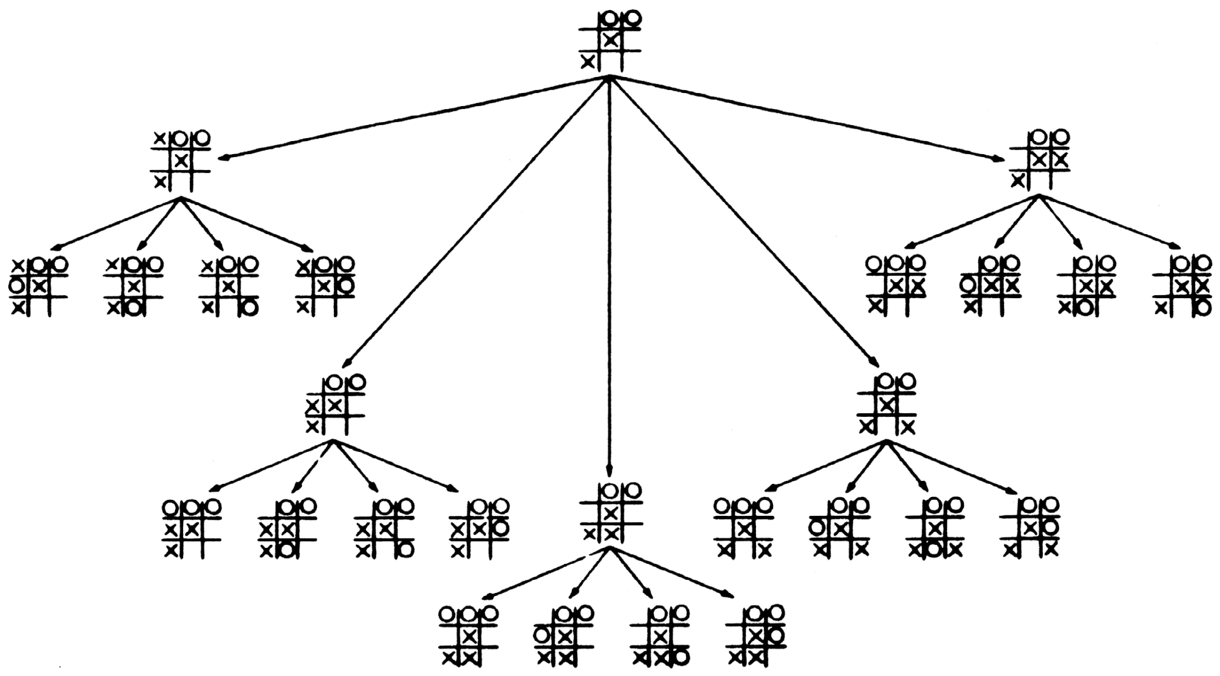

Figure 2. Alternative Futures in Tic-Tac-Toe

IV. Reality as Fantasy

The *stuff* of computer science, symbol structures and sets of instructions, doesn't seem to be as real as the stuff of other scientific disciplines. But aren't other scientists and engineers living in a bit of a dream world also? What do we mean by *real* anyway? In computer science, we are *forced* to admit that symbol structures are merely inventions—vocabulary useful to help us think about and design our computer systems. The arbitrariness of this vocabulary is obvious in computer science, but aren't the concepts of physics and chemistry *arbitrary* also? Does a molecule *really* exist (whatever that means) or is it, too, just a *story* concocted by physicists to help us understand the world? In computer science, we have no choice but to be honest and to admit that we deal in inventions. Perhaps this view will prove useful to other philosophers of science.

Because the notion of complex structures composed of layers of simpler structures is so prominent in computer science, we, perhaps more than others, easily accept the fact that our very conception of reality is one of *layers* of increasingly abstract *fictions*. To be more concrete about this idea, let's consider the problem of controlling an elevator from the viewpoints of electrical engineering and computer science. Thought about in terms of basic electronics, we can imagine currents that are initiated by button-pressing people on different floors of a building. Depending on the flow of currents from the various floors, electronic circuitry controlling the elevator makes it go up, down, or stop, and opens and closes the elevator doors. The diagram describing the detailed electronic circuits that control a bank of elevators in a tall office building would be rather complex. At a slightly more abstract level, we can imagine a somewhat simpler diagram of on-off signalling switches controlling the speed of the elevator motors. Moving higher, now solidly into the realm of computer science, we can describe the system by a computational model of instructions that change a symbol structure representing the states of the elevators. Even higher levels of explanation would refer to the requests of passengers to go up or down.

Let's get on a metaphorical elevator near the top level of explanation and take it down to see if we ever find the *ground floor of reality*. We could say that a description in terms of *requests* doesn't actually mirror *reality* —what's *real* are the *data structures*, *lists*, and *subroutines* on the floor below. Yet, these things aren't *real* either. The newcomer to computer science learns that what *really exists* are *bits*—patterns of *on* and *off* transistors in the computer hardware. Yet, the more sophisticated realize that *on-ness* and *off-ness* are fictional constructs also—constructs that help us explain the relevant aspects of the workings of electrical phenomena. But these electrical ideas are fictions also—invented to help us explain semi-conductor physics, which—in turn—uses its own fictions to describe what happens in certain crystal structures. The layers of fiction never stop—an endless descent of explanations to help us predict phenomena of ever increasing detail. Science never reaches the ground floor—we can always invent floors below. Ever more imaginative fiction is as close as we can get to reality! Even our everyday, commonsense understanding of rocks and mountains, trees and forests has to be artfully constructed in our minds.

Even some scientists and engineers need to be reminded occasionally that their *models* of the world are not the same as the world itself. Models are fictional constructs invented by scientists to describe the world. Whenever they notice that their models do not adequately predict the results of their experiments, they must invent better fiction. Sometimes these constructions are just minor adjustments to the old story; sometimes a completely new story (called a *paradigm shift*) is needed. Always, these stories are composed of symbols—the very stuff of computer science! In this sense, computer science is even more basic than physics!

Of course this view of the relation between descriptions of reality and reality itself profoundly affects our understanding of the nature of science. Jacob Bronowski wrote that "The most remarkable discovery made by science is science itself."[3] But I don't think science has finished discovering science yet, because we don't know all we need to know about the nature of knowledge and how it is that a system can know about its world. We will truly understand these things only when we ourselves can build systems that know about their worlds. In the words of the seventeenth-century Italian philosopher Vico, *"Certum quod factum."* (We are certain only of what we build.)

Perhaps these layers of explanation tell us more about how the human brain works than they do about the world itself. In order for us humans to comprehend our world, we must organize our story into these levels. Engineers, no less than other humans, must live by this same habit. In order for them to design complex artifacts, they must understand their design in layers. The discipline called *structured programming* is just the rediscovery by computer scientists of what all engineers have had to learn, namely that design must occur in levels.

Failure to appreciate the idea of levels can lead to serious misunderstandings. Some scientists think that knowing more thoroughly how the *neuron* works will lead to a better understanding of how the *mind* works. But most engineers will guess that *however* the neuron works is irrelevant to any useful description of the mind—just as how a transistor works is really irrelevant to most of computer science. Neurons and transistors are insulated by too many intermediate levels of explanation from minds and complex computations for there to be any important residual connections between them. Bronowski recognized this fact when he stated "What is to be said about [the principles of the brain] then, comes not from physics and chemistry and biology, but from symbolic logic."[4]

V. Putting it All Together

To summarize what we have said so far, computer scientists are concerned with three kinds of operations in which symbol structures play a role. First, we are interested in how symbol structures (in the form of instructions like those used in the Jacquard loom and the player piano) can be used to affect the physical world. This use is critical in such modern applications as numerically controlled machine tools. We can think of these processes as a bridge or connection from fantasy to reality. Secondly, we are concerned with how symbol structures can be used to change other symbol structures. The elaboration of this subject forms much of the content of computer science itself and accounts for the real power of the computer. We could say that this bridge connects fantasy to other fantasy. Third, we want to let the world in to our symbol structures by having them be affected in appropriate ways by signals from specialized sensors. This topic engages us in the study of perception and would permit us ultimately to build ears that could understand human speech and eyes that could see—bridges connecting external reality to an inner fantasy of symbols.

If we combine all three kinds of bridges, we get machines that perceive, reason and act—in short robots! Today's robots, though useful and perhaps even frightening, are nowhere near as subtle and versatile as they will be tomorrow. Already, this three-bridge technology is used to guide subway trains, steer cruise missiles, operate nuclear power stations, and build automobiles. Of more important intellectual interest, these systems, with their perceptual and effector systems as well as their ability to manipulate internal symbol structures, provide biologists and psychologists with their best models yet for explaining how animals behave.

Will our study of computation ultimately lead us to an understanding of the human mind including its "creative abilities"? Bronowski, on the one hand seems pessimistic (although some might call it optimistic) about this point. Speaking about the creative process, he said "It is a free play of the mind, an invention outside the logical processes...

[3] Jacob Bronowski, "The Creative Process," in *A Sense of the Future*, The MIT Press, Cambridge, MA, 1977 (page 6).

[4] Jacob Bronowski, "The Logic of the Mind," *op. cit.* (page 58).

an unmechanical act of free choice."[5] He was a sufficiently careful scientist though also to say (in 1965) that, "[In] the measureless future, [this question] cannot be answered. A machine is not a natural object; it is a human artifact which mimics and exploits our own understanding of nature; and we cannot foresee how radically we may come to change that understanding. We cannot foresee and we cannot conceive all possible machines."[6] My bet is that the measureless future is going to unfold a bit faster than even Bronowski might have guessed.

VI. Acknowledgements

Karen Nilsson, Terry Winograd, Robert Eustis, and Derek Sleeman made suggestions for improving the clarity and accuracy of this essay. I adopted many of them.

[5] Jacob Bronowski, "The Logic of the Mind," op. cit. (pages 62-63).
[6] Jacob Bronowski, "The Identity of Man," as quoted in "The Logic of Mind," op. cit. (pages 72-73).