CMU-CS-80-110

Physical Symbol Systems

Allen Newell

March 1980

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Paper given at the La Jolla Conference on Cognitive Science, 15 Aug 1979.

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78-C-1551.

ABSTRACT

Allen Newell, Carnegie-Mellon University

On the occasion of a first conference on Cognitive Science, it seems appropriate to review the basis of common understanding between the various disciplines. In my estimate, the most fundamental contribution so far of artificial intelligence and computer science to the joint enterprise of cognitive science has been the notion of a *physical symbol system*, ie, the concept of a broad class of systems capable of having and manipulating symbols, yet realizable in the physical universe. The notion of *symbol* so defined is internal to this concept, so it becomes a hypothosis that this notion of symbols includes the symbols that we humans use everyday of our lives. In this paper we attempt systematically, but plainly, to layout the nature of physical symbol systems. Such a review is in ways familiar, but not thereby useless. Restatement of fundamentals is an important excercise.

****

Table of Contents

List of Figures

PHYSICAL SYMBOL SYSTEMS[1]

1. INTRODUCTION

The enterprise to understand the nature of mind and intelligence has been with us for a long time. It belongs not to us alone, who are gathered at this conference, nor even to science alone. It is one of the truly great mysteries and the weight of scholarship devoted to it over the centuries seems on occasion so oppressively large as to deny the possibility of fundamental progress, not to speak of solution.

Yet, for almost a quarter century now, experimental psychology, linguistics and artificial intelligence have been engaged in a joint attack on this mystery that is fueled by a common core of highly novel theoretical ideas, experimental techniques and methodological approaches. Though retaining our separate disciplinary identities, we have strongly influenced each other throughout this period. Others have been involved in this new attack, though not so centrally -- additional parts of computer science and psychology, and parts of philosophy, neurophysiology and anthropology.

Our communality continues to increase. In consequence, we are engaged in an attempt to bind our joint enterprise even more tightly by a common umbrella name, *Cognitive Science*, a new society, and a new series of conferences devoted to the common theme -- the outward and institutional signs of inward and conceptual progress. On such an occasion, attempts to increase our basis of mutual understanding seem to be called for.

In my own estimation (Newell & Simon, 1976), the most fundamental contribution so far of artificial intelligence and computer science to this joint enterprise has been the notion of a *physical symbol system*. This concept of a broad class of systems that is capable of having and manipulating symbols, yet is also realizable within our physical universe, has emerged from our growing experience and analysis of the computer and how to program it to perform intellectual and perceptual tasks. The notion of *symbol* that it defines is internal to this concept of a system. Thus, it is an hypothesis that these symbols are in fact the same symbols that we humans have and use everyday of our lives. Stated another way, the hypothesis is that humans are instances of physical symbol systems, and by virtue of this mind enters into the physical universe.

In my own view this hypothesis sets the terms on which we search for a scientific theory of mind.

---

[1]Herb Simon would be a co-author of this paper, except that he is giving his own paper at this conference. The key ideas are entirely joint, as the references indicate. In addition, I am grateful to Greg Harris, John McDermott, Zenon Pylyshyn and Mike Rychener for detailed comments on an earlier draft.

What we all seek are the further specifications of physical symbol systems that constitute the human mind or that constitute systems of powerful and efficient intelligence. The physical symbol system is to our enterprise what the theory of evolution is to all biology, the cell doctrine to cellular biology, the notion of germs to the scientific concept of disease, the notion of tectonic plates to structural geology.

The concept of a physical symbol system is familiar in some fashion to everyone engaged in Cognitive Science. Familiar, yet perhaps not fully appreciated. For one thing, this concept has not followed the usual path of scientific creation, where development occurs entirely within the scientific attempt to understand a given phenomenon. It was not put forward at any point in time as a new striking hypothesis about the mind, to be confirmed or disconfirmed. Rather, it has evolved through a much more circuitous root. Its early history lies within the formalization of logic, where the emphasis was precisely on separating formal aspects from psychological aspects. Its mediate history lies within the development of general purpose digital computers, being thereby embedded in the instrumental, the industrial, the commercial and the artificial -- hardly the breeding ground for a theory to cover what is most sublime in human thought. The resulting ambivalence no doubt accounts in part for a widespread proclivity to emphasize the role of the *computer metaphor* rather than a *theory of information processing*.

The notion of symbol permeates thinking about mind, well beyond attempts at scientific understanding. Philosophy, linguistics, literature, the arts -- all have independent and extensive concerns that focus on human symbols and symbolic activity. Think only of Cassirier or Langer or Whitehead, in philosophy. Consider semantics, concerned directly with the relation between linguistic symbols and what they denote. Or Jung, in a part of psychology remote from experimentation and tight theory. These are vast realms of scholarship, by any reckoning.

I cannot touch these realms today in any adequate way. Perhaps, I can let one quote from Whitehead stand for them all:

> "After this preliminary explanation we must start with a definition of symbolism: The human mind is functioning symbolically when some components of its experience elicit consciousness, beliefs, emotions, and usages, respecting other components of its experience. The former set of components are the 'symbols', and the latter set constitute the 'meaning' of the symbols. The organic functioning whereby there is transition from the symbol to the meaning will be called 'symbolic reference'." (Whitehead, 1927, pp7-8)

This statement, from over fifty years ago, has much to recommend it. Let it serve as a reminder that the understanding of symbols and symbolism is by no means brand new. Yet the thread through computer science and artificial intelligence has made a distinctive contribution to discovering the nature of human symbols. Indeed, in my view the contribution has been decisive.

The notion of a physical symbol system has been emerging throughout the quarter century of our joint enterprise -- always important, always recognized, but always slightly out of focus as the decisive scientific hypothesis that it has now emerged to be.

For instance, recall the rhetoric of the fifties, where we insisted that computers were *symbol manipulation machines* and not just *number manipulation machines*. The mathematicians and engineers then responsible for computers insisted that computers only processed *numbers* -- that the great thing was that instructions could be translated into numbers. On the contrary, we argued, the great thing was that computers could take instructions and it was incidental, though useful, that they dealt with numbers. It was the same fundamental point about symbols, but our aim was to revise opinions about the computer, not about the nature of mind.

Another instance is our ambivalence toward list processing languages. Historically, these have been critically important in abstracting the concept of symbol processing and we have certainly recognized them as carriers of theoretical notions. Yet we have also seen them as *nothing but* programming languages, ie, as nothing but tools. The reason why AI programming continues to be done almost exclusively in list processing languages is sought in terms of ease of programming, interactive style and what not. That Lisp is a close approximation to a pure symbol system is often not accorded the weight it deserves.

Yet a third instance can be taken from our own work. When we laid out the notion of physical symbol system in our book on human problem solving (Newell & Simon, 1972), we did this as an act of preparation, not as the main point. We focussed the theory on how people solved problems, given that they were symbol manipulation systems. Even when, a little later, we chose to focus on the physical symbol system hypothesis per se (Newell & Simon, 1976), it was in the context of receiving an award and thus we described it as a conceptual advance that had already transpired.

A fourth and final instance is the way information processing systems are presented in cognitive psychology. Even in the best informed presentations (eg, Clark & Clark, 1977; Lindsay & Norman, 1977, Rumelhart, 1977) there is little emphasis on symbolic functioning per se. When concern is expressed about the adequacy of information processing notions for psychology (eg, Neisser, 1976) , the role of symbolic functioning is not addressed. There are some very recent exceptions to this picture (Lachman, Lachman & Butterfield, 1979). But some of these (Allport, 1979, Palmer, 1978) seem to view such developments as rather new, whereas I see them as having been the taproot of the success in Artificial Intelligence right from the start almost 25 years ago.

In sum, it seems to me, a suitable topic for this conference is to attempt, systematically but plainly, to lay out again the nature of physical symbol systems. All this will be in some ways familiar, but I hope far from useless. Restatement of fundamentals is an important exercise. Indeed, I can take my

text from Richard Feynman. He is speaking of Fermi's law of optics, but it applies generally:

> "Now in the further development of science, we want more than just a formula. First we have an observation, then we have numbers that we measure, then have a law which summarizes all the numbers. But the real *glory* of science is that we can find a way of thinking such that the law is *evident*." (Feynman, 1963, p26)

Physical symbol systems are becoming for us simply *evident*. But they are our *glory*, and it is fitting that we should understand them with a piercing clarity.

And so, if you cannot stand what I say here as science, then take it as celebration.

1.1. Constraints on Mind

Let me provide a general frame for the paper. The phenomena of mind have arisen from a complex of aspects of the physical universe, localized strikingly (though possibly not exclusively) in us humans. We scientists, trying to discern the physical nature of mind, can cast these aspects as a conjunction of constraints on the nature of mind-like systems. Then our discovery problem is that of finding a system structure that satisfies all these constraints. In trying to make that discovery, we can use any tactics we wish. The constraints themselves are simply desiderata and have no privileged status.

There is no magic list of constraints that we can feel sure about. Their choice and formulation is as much a step in the discovery process as solving the constraint satisfaction problem after positing them. However, it is easy to list some candidate constraints that would find general acknowledgement. Figure 1-1 presents a baker's dozen.

These constraints are far from precisely defined. Operationalizing the notion of self-awareness poses difficult problems, however critical it seems as a requirement. Even what constitutes the brain is open, moving over the last thirty years from an essentially neural view to one that includes macromolecular mechanisms as well. Not all the constraints are necessarily distinct. Conceivably, human symbolic behavior and linguistic behavior could be the same, as could development and learning. Not all constraints are necessarily independent. To be a neural system implies being a physical system, though there can be reasons to consider the more general constraint separately. Some of the constraints are familiar back to Aristotle. Others are recent additions. Who would have thought to add the concern with robustness under error, if computers and their programs had not exhibited the sort of brittle, ungraceful degradation that we have all come to know so well.

What seems clear is that, when we finally come to know the nature of mind in humans, it will be seen to satisfy all of these constraints (and others that I have neglected to list). And when we finally come to know the nature of intelligence generally, it will be seen how its variety arises from a release

1. Behave as an (almost) arbitrary function of the environment (universality).
2. Operate in real time.
3. Exhibit rational, ie, effective adaptive behavior.
4. Use vast amounts of knowledge about the environment.
5. Behave robustly in the face of error, the unexpected, and the unknown.
6. Use symbols (and abstractions).
7. Use (natural) language.
8. Exhibit self-awareness and a sense of self.
9. Learn from its environment.
10. Acquire its capabilities through development.
11. Arise through evolution.
12. Be realizable within the brain as a physical system.
13. Be realizable as a physical system.

Figure 1-1: Constraints on Mind.

from some of these constraints.

Our difficulty, as scientists, is that we cannot solve for systems that satisfy such simultaneous constraints. Indeed, we cannot usually do better than to generate on one constraint and test on the others. Thus, particular constraints are taken by various groups of scientists as the frame within which to search for the nature of mind. One thinks of Ashby (1956) and his formulation in terms of general differential equation systems, which is to say, basically physically realizable systems. Or the endeavor of those working in the fifties on self-organizing systems to work within neuron-like systems (Yovits & Cameron, 1960, Yovits, Jacobi & Goldstein, 1962). Or the emergence of a sociobiology that works primarily from evolutionary arguments (Wilson, 1975). And, of course, both the neurophysiologists and the linguists essentially work from within their respective disciplines, which correspond to constraints in our list. Artificial intelligence works from within the digital computer, sometimes it seems even from within Lisp. However, the computer is not one of these constraints, though strongly associated with the first item, and my purpose is not to identify particular constraints with particular disciplines. The constraints are conceptual aspects of the nature of human mind, and they must all be taken into account in the final analysis, whatever the starting point.

Which constraint forms a preferred basis from which to conduct the search for the nature of mind? Most important, a constraint must provide a *constructive* definition of a class of systems. Otherwise, search within it cannot occur, because it will not be possible to generate candidates whose properties can then be explored. Several of the constraints have real difficulty here -- development and learning, robustness, real-time operation. For instance, we simply have no characterization of all systems that show development; all we can do is pose a system described within some other class and ask about its developmental characteristics. The constraint of development must remain primarily a test, not a generator. On the other hand some constraints, such as using language, do very well. The formalisms for grammars provide potent generative bases.

The strength of a constraint, or its distinctiveness with respect to mind, also weighs in the balance, however difficult the assessment of such a characteristic. For example, one real problem with the evolution constraint is that we know it gives rise to an immense diversity of systems (organisms). It is not clear how to get it to generate systems that are shaped at all to mind-like behavior. Again linguistics has fared much better in this regard. For linguistics has appeared, until recently, to be distinctively and uniquely human. As a last example, one major argument against the universal machines of logic and computer science has always been that universality had been purchased at the price of total inefficiency, and a class which relied on such an aspect seemed irrelevant to real systems.

But such considerations are only preferences. Our joint scientific enterprise demands that

substantial groups of scientists focus on all these constraints, and their various combinations. It demands that new constraints be discovered and added to the list, to provide new ways from which to seek the true nature of mind.

My focus on physical symbol systems in this paper certainly amounts to an argument for taking one particular class of systems as the base -- as the generator -- for the search for mind. This class appears to satisfy jointly at least two of the constraints in the list -- *universality* and *symbolic behavior* -- and to give good evidence of being able to be shaped to satisfy other constraints as well, while still remaining usefully generative. But, as the discussion should make clear, this is just an argument over scientific tactics -- over the right way to go about untying the great puzzle knot that is the mind. On the matter of scientific substance, we need to understand all we can about all the aspects represented in these constraints.

1.2. Plan

Let me preview what I intend to do, so as to be as plain and straightforward as possible.

To present the notion of a physical symbol system, I introduce a specific example system. This permits a concrete discussion of the key property of universality, the first constraint on our list. With this concept in hand, I generalize the example system to the class of all physical symbol systems. This makes evident that systems that satisfy the constraint of universality also are capable of a form of symbolic behavior. The Physical Symbol System Hypothesis states in essence that this form of symbolic behavior is all there is; in particular, that it includes human symbolic behavior. I turn briefly to the question of system levels, which allows the placement of the symbol level within the larger frame of physical systems. With all these elements on the table, I then discuss some issues that are important to understanding the notion of physical symbol system and the hypothesis, and their role in cognitive science.

So far I have been careful always to refer to a *physical* symbol system, in order to emphasize two facts. First, such a system is realizable in our physical universe. Second, its notion of symbol is a priori distinct from the notion of symbol that has arisen in describing directly human linguistic, artistic and social activities. Having been clear about both of these, we can drop the adjective, except when required to emphasize these two points.

As already stated, the fundamental notion of a physical symbol system presented here is not novel scientifically. Even the formulation presented does not differ in any important way from some earlier attempts (Newell & Simon, 1972, Newell & Simon, 1976). I am engaged in restatement and explication. The details and the tactics of the formulation are new and I hope thereby to make matters exceptionally clear and to highlight some important features of such systems. Still, it does not follow

that the notion of physical symbol system and the particular hypothesis about it are accepted by all, or accepted in exactly the form that is given here.

2. SS: A PARADIGMATIC SYMBOL SYSTEM

Figure 2-1 lays out our example symbol system schematically. We will call it *SS* (Symbol System) for short. It is a machine which exists in an environment consisting of *objects*, distributed in a space of *locations*. We can imagine the objects having some sort of structure and dynamics, and doing their individual and interactive thing in this space.

SS consists of a *memory*, a set of *operators*, a *control*, an *input* and an *output*. Its inputs are the objects in certain locations; its outputs are the modification or creation of the objects in certain (usually different) locations. Its external behavior, then, consists of the outputs it produces as a function of its inputs. The larger system of environment plus SS forms a closed system, since the output objects either become or affect later input objects. SS's internal state consists of the state of its memory and the state of the control; and its internal behavior consists of the variation in this internal state over time.

The memory is composed of a set of *symbol structures*, $\{E_1, E_2, .. E_m\}$, which vary in number and content over time. The term *expression* is used interchangeably with *symbol structure*. To define the symbol structures there is given a set of abstract *symbols*, $\{S_1, S_2, ..., S_n\}$. Each symbol structure is of a given *type* and has some number of distinguished *roles*, $\{R_1, R_2, ...\}$. Each role contains a symbol, so if we take the type as understood implicitly and the roles as the successive positions on the paper, we could write an expression as:

$[S_1\ S_2 \ldots\ S_n]$

If we wanted to show the roles and the type explicitly we could write:

$[\text{Type:T}\ R_1{:}S_1\ R_2{:}S_2 \ldots\ R_n{:}S_n]$

The roles (and their number) are determined by the *type*, of which there can be a large variety. The same symbol, eg, S_k, can occupy more than one role in a structure and can occur in more than one structure. By the *content* of an expression is meant simply the symbols associated with the roles of the expression.

SS has ten operators, each shown as a separate box in the figure. Each operator is a machine that takes one or more symbols as input and produces as output some symbol (plus possibly other effects) as a result. The behavior that occurs when an operator and its inputs combine is called an *operation*. The details of the behavior of the system comes from these operations, which we will go over in a moment.

The behavior of the system is governed by the control. This is also a machine. Its inputs include the operators: it has access to their inputs and outputs, and can evoke them. It also has as an input the symbol for a single expression, which is called the *active* expression. The behavior of the control

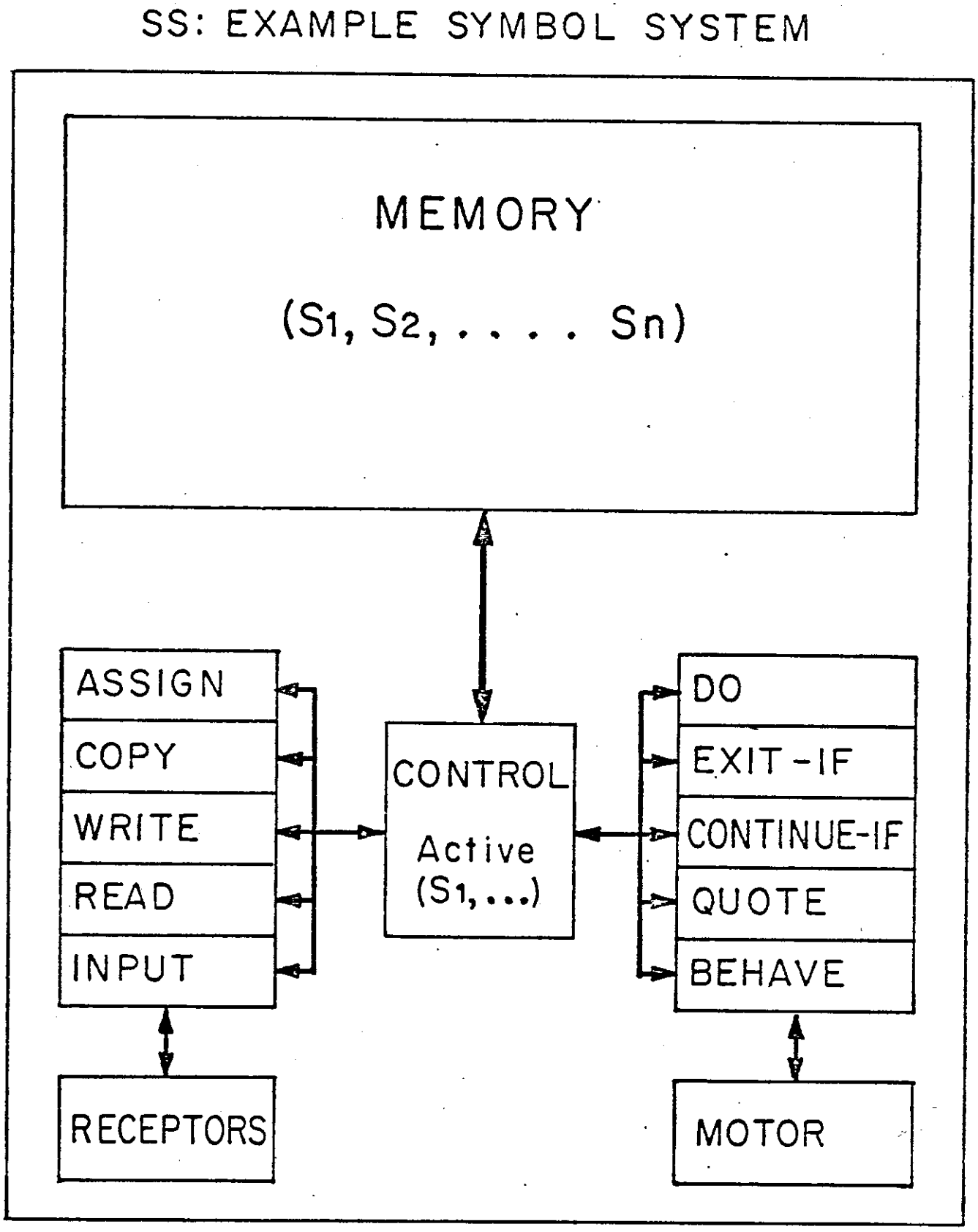


Figure 2-1: Structure of SS, a paradigmatic symbol system

consists of the continual *interpretation* of whatever expression is active. If this specifies an operation to be performed, then the control will bring the input symbols to the input locations of the indicated operator and then evoke the operator to produce the result, ie, it will effect the combining of data and operators. The control also determines which expression shall become active next, so that the behavior of the total system runs on indefinitely. We will describe the control in detail after discussing the operators.

Figure 2-2 lists the operations of SS. There exists a type of symbol structure, which we will call a *program*, which has roles corresponding to an operator and the inputs appropriate to that operator. These program expressions are shown in the figure at the right.

- **Assign** a symbol. This establishes a basic relationship between a symbol and the entity to which it is assigned, which we call *access*. While it lasts (ie, until the assignment is changed), any machine (ie, the ten operators and the control) that has access to an occurrence of this symbol in an expression has access to the assigned entity. If a machine has access to an expression, then it can obtain the symbols in the various roles of the expression and it can change the symbols in these roles. Symbols can be **assigned** to entities other than expressions, namely, to operators and roles. Access to an operator implies access to its inputs, outputs and evocation mechanism. Access to a role of a given type implies access to the symbol at that role for any expression of the given type and access to write a new symbol at that role.

- **Copy** expression. This adds expressions and symbols to the system. The new expression is an exact replica of the input expression, ie, the same type and the same symbols in each role. A new symbol is created along with the new expression (a necessity for gaining access to the expression).

- **Write** an expression. This creates expressions of any specified content. It does not create a new expression (**copy** does that), but modifies its input expression. What to **write** is specified by giving the roles and the new symbols that are to occupy these roles. **Write** permits several symbols to be written with a single operation; it could as well have permitted only one. For example, given a type with roles R_1, R_2, etc., in order, and given an expression (X Y Z), (**write** (X Y Z) R_1 A R_3 C) produces a modified expression (A Y C). **Write** establishes a symbol at a given role whether or not there was a symbol at that role before, and independent of what symbols exist at other roles. **Writing** nil at a role effectively deletes it.

- **Read** the symbol at a specific role. This obtains the symbols that comprise an expression, given that the expression has been obtained. It is possible that no symbol exists for a given role; in this case **read** produces the symbol nil. (Thus it can be seen why **writing** nil at a role effectively deletes it.)

- **Do** sequence. This makes the system do arbitrary actions, by specifying that it do one thing after another. There are an unlimited number of input roles, one for each element in the sequence. The last expression produced during such a sequence is taken to be the result of the sequence. All the expressions produced by earlier items in the sequence are ignored. Of course, actions may have taken place along the way (often referred to as side effects), eg, assignment of symbols.

Assign symbol S_1 to the same entity as symbol S_2 Produces S_1 with new assignment	**(assign S_1 S_2)**
Copy expression E (create new symbol) Produces newly created expression and symbol	**(copy E)**
Write S_1 at role R_1, ..., in expression E Produces the modified expression nil is the same as doesn't exist	**(write E R_1 S_1 ...)**
Read symbol at role R of E Produces the expression or nil	**(read R E)**
Do sequence S_1 S_2 S_3 ... Produces the expression produced by last S_i	**(do S_1 S_2 ...)**
Exit sequence if the prior result is E Produces prior expression	**(exit-if E)**
Continue sequence if the prior result is E Produces prior expression	**(continue-if E)**
Quote the symbol S Produces S without interpretation	**(quote S)**
Behave externally according to expression E. Produces feedback expression	**(behave E)**
Input according to expression E Produces new expression or nil	**(input E)**

Figure 2-2: Operators of SS

- **Exit-if** and **Continue-if**. The system behaves conditionally by continuing or exiting (terminating) the execution of a sequence. A conditional operator tests if the expression produced at the immediately preceding step of the sequence is the same as its input expression. It then takes a specific control action. For example, (**do** ... A (**exit-if** A) ...) would exit, ie, would not complete the rest of the sequence. If symbols A and B designate different expressions, then (**do** ... B (**continue-if** A) ...) would also exit. The output of the operator is the expression tested, which then becomes the output of the sequence if there is termination.

- **Quote** a symbol. The control automatically interprets every expression that becomes active. This operator permits it to not interpret a given expression, but to treat its symbol as the final result.

- **Behave** externally. There exists some collection of external behaviors controllable by SS. Symbol structures of some type exist that instruct the organs that produce this external behavior. It will be enough to have an operator that evokes these expressions. Execution of the operator will produce some expression that provides feedback about the successful accomplishment (or failure) of the external operation.

- **Input** from environment. Inputs from the external environment enter the system by means of newly created expressions that come to reside in the memory. These inputs occur when the **input** operator is evoked; there may be different channels and styles of input, so that **input** takes an expression as input to specify this. The input expressions are processed when **input** is evoked, since the resulting expression is interpreted by the control, though presumably the new expressions are not of type program, but some type related to describing the external environment.

The operation of the control is shown in Figure 2-3. The control continuously interprets the active expression. The result of each interpretation is ultimately a symbol, though other actions (ie, side effects) may have occurred during the act of interpretation, which are also properly part of the interpretation.

Control interprets the active expression by first determining whether it is a program symbol structure. Thus the control can sense a structure's type. If it is not a program, then the result of the interpretation is just the symbol itself (ie, the symbol is treated as data).

If the active expression is a program, then the control proceeds to execute the operation specified by the program. However, the actual symbols in the program at the roles for the operator and its inputs must themselves be interpreted. For these symbols might not be the operator and inputs, respectively, but programs whose interpretations are these symbols. Thus, the control interprets each symbol in the program, until it finally obtains the actual symbols to be used for the operator and the inputs. Then, it can actually get the operation performed by sending the input symbols to the appropriate operator, evoking it, and getting back the result that the operator produces.

Control then interprets the result (as arrived at through either of the routes above). If it is a new

Interpret the active expression:

If it is not a program:
Then the result is the expression itself.

If it is a program:
Interpret the symbol of each role for that role;
Then execute the operator on its inputs;
Then the result of the operation is the result.

Interpret the result:

If it is a new expression:
Then interpret it for the same role.

If it is not a new expression:
Then use as symbol for role.

Figure 2-3: Operation of SS's Control

expression, then it proceeds to interpret it. If it is not new, then it finally has obtained the symbol.

The control has the necessary internal machinery to interpret each operator or input symbol in a program until it obtains the symbol finally to be used for each role in the program. This will be the one that is finally not a program type of structure. The control remembers the pending interpretations and the results produced so far that are still waiting to be used. The normal way to realize all this in current technology is with a pushdown stack of contexts; but all that is specified here is end result of interpretation, not how it is to be accomplished.

We now have an essentially complete description of one particular symbol system. To generate a concrete (and particular) behavioral trajectory, it is only necessary to provide an *initial* condition, consisting of the set of initial expressions in the memory and the initial active expression. The system behaves in interaction with the environment, but this is accounted for entirely by the operation of the **input** and **behave** operators. The operation of these two operators depends on the total environment in which the system is embedded. They would normally be given by definite mechanisms in the external structure of the system and the environment, along with a set of laws of behavior for the environment that would close the loop between output and input. From a formal viewpoint the operation of these two operators can just be taken as given, providing in effect a *boundary* condition for the internal behavior of the system.

This sort of a machine is certainly familiar to almost everyone in Cognitive Science, at least in outline. The virtue of SS, over others that might be even more familiar, is that it is designed to aid understanding the essential features of symbols and symbolic behavior. There are no irrelevant details of SS's structure. Each operator (and also the control) embodies a generalized function that is important to understanding symbolic systems.

The expository virtues of SS aside, it remains a garden variety, Lisp-ish sort of beast.

3. UNIVERSALITY

That our example symbol system is garden variety does not keep it from being a variety of a very remarkable genus. Symbol systems form a class. It is a class that is characterized by the property of *universality*. We must understand this remarkable property before we can generalize appropriately from our paradigmatic symbol system to a characterization of the entire class.

Central to universality is flexibility of behavior. However, it is not enough just to produce any output behavior; the behavior must be *responsive* to the inputs. Thus, a universal machine is one that can produce an arbitrary input-output function, that is, can produce any dependence of output on input.

Such a property is desirable for an adaptive, intelligent system, which must cope with environments whose demands are not known at the time the system is designed. Indeed, this property heads the constraints in Figure 1-1. Being able to produce *any* behavior in response to a situation is neither absolutely necessary nor hardly sufficient for success. But the more flexibility the better; and if behavior is too unresponsive, the system will fail against its environment. Almost all purposive behavior shows intricate dependence on the environment, ie, shows the flexible construction of novel input-output functions -- an animal circling its prey, a person in conversation with another, a player choosing a chess move, a student solving a physics exercise, a shopper bargaining with a seller, and on and on. This was the classic insight of Cybernetics -- systems appeared *purposive* when their behavior was dependent on the environment so as to attain (or maintain) a relationship; and *feedback* was necessary to obtain this dependence with a changing environment. The formulation here separates the ability to produce the dependence (universality) from the way such an ability can be used to produce purposiveness, the latter residing in the rationality constraint in Figure 1-1.

The property of universality cannot be quite so simply defined. Four difficulties, in particular, must be dealt with.

The first difficulty is the most obvious. Any machine is a prisoner of its input and output domains. SS, our example system, presents an abstract machine-centered view, so that the external world is pretty much what is seen by the machine. But this is deceptive. Machines live in the real world and have only a limited contact with it. Any machine, no matter how universal, that has no ears (so to speak) will not hear; that has no wings, will not fly. Thus universality will be relative to the input and output channels. Such a consideration is alleviated in theoretical discussions by the simple expedient of considering only abstract inputs and ouputs. It can be alleviated in the real world by providing transducers that encode from one input-output channel to another. Thus, being able to produce any function between two given domains, permits inducing any function between two other domains, if the

domains are hooked up appropriately.[2] But this interface limit must always be remembered.

The second difficulty is also obvious. In the physical world there are limits -- limits to the speed of components, to spatial and energy sensitivity, to material available for memory, to reliability of operation, to name just the more obvious. To state a tautology: No system can behave beyond its physical limits. Thus, the universality of any system must be taken relative to such physical implementation limits.

The third difficulty is more serious. A machine is defined to be a system that has a specific determined behavior as a function of its input. By definition, therefore, it is not possible for a single machine to obtain even *two* different behaviors, much less any behavior. The solution adopted is to decompose the input into two parts (or aspects): one part (the *instruction*) being taken to determine which input-output function is to be exhibited by the second part (the *input-proper*) along with the output. This decomposition can be done in any fashion, for instance, by a separate input channel or by time (input prior to a starting signal being instruction, afterward being input-proper). This seems like an innocent arrangement, especially since the input-proper may still be as open as desired (eg, all future behavior). However, it constitutes a genuine limitation on the structure of the system. For instance, the instruction must have enough capacity to specify all of the alternative functions. (Eg, if the instruction to a machine consists only of the setting of a single binary toggle switch, then the machine cannot exhibit three different input-output behaviors.) Most important, the basic decomposition into two parts has far reaching consequences -- it guarantees the existence of symbols.

The fourth difficulty is the most serious of all. There appears to be no way that a universal machine can behave literally according to *any* input-output function, if the time over which the behavior is to occur is indefinitely extended (eg, the entire future after some period of instruction). This is the import of the discovery of *non-computable* functions, which is an important chapter in the theory of computing machines (Minsky, 1967, Brainerd & Landweber, 1974). The difficulty is fundamentally that there are too many functions -- too many ways to have to instruct a machine to behave.

This can be appreciated directly by noting that each instruction to the machine, no matter how complex, is simply a way of naming a behavior. Thus, a machine cannot produce more distinct behaviors than it can have distinct instructions. Let the number of possible instructions be K. The number of behaviors is the number of input-ouput functions; so if there are M possible inputs and N possible outputs, then the number of behaviors is N^M (ie, the assignment of one of the N possible

[2]The internal domains must have enough elements to permit discrimination of the elements of the external domains, a condition which Ashby (1956) called the *Law of requisite variety*.

outputs for each of the M inputs). Thus, K instructions must label N^M behaviors. If K, M and N are all in the same range, then N^M is going to be *very* much bigger than K. Now, as time is permitted to extend indefinitely into the future, all three possibilities (K, M and N) will grow to become countably infinite. But, though K (the number of instructions) grows to be countably infinite, N^M (the number of functions to be labeled) grows much faster to become uncountably infinite. In sum, there simply are not enough possible instructions to cover all the functions that must be named.

If all possible functions cannot be attained, then some way must be found to describe which can and which cannot. Therein lies a further difficulty. Suppose a descriptive scheme of some sort is used, in order to say that a given machine can realize functions of certain descriptions and not functions of other descriptions. What do we know then about the functions that are not describable by the given scheme? We have confounded the properties of the descriptive scheme with the properties of the machine. Indeed, the suspicion might arise that a connection exists between descriptive schemes and machines, so that this difficulty is part and parcel of the main problem itself.

The solution has been to take the notion of a machine itself as the keystone. Direct description of behavior is abandoned, and in its place is put *the behavior produced by such and such a machine*. For any class of machines, defined by some way of describing its operational structure, a machine of that class is defined to be universal if it can behave like any machine of the class. This puts simulation at the center of the stage; for to show a given input-output behavior is to simulate a machine that shows that input-output behavior. The instructional input to the machine must now be some means of describing any arbitrary machine of the given class. The machine whose universality is being demonstrated must take that input and behave identically to the machine described by its input, ie, it must simulate the given machine.

The notion of universality thus arrived at is *relative*, referring only to the given class of machines. Universal machines could exist for classes of machines, all right, but the input-output functions ecompassed by the whole class could still be very limited. Such a universal machine would be a big frog in a small pond of functions.

The next step is to attempt to formulate very large classes of machines, by means of general notions of mechanism, in order to encompass as wide a range of input-output functions as possible. (The input and output domains are always taken to be intertranslatable, so the relevant issue is the functional dependence of output on input, not the character of the inputs and outputs taken separately.) Another important chapter in the theory of computing (Minsky, 1967, Brainerd & Landweber, 1974) has shown that all attempts to do this lead to classes of machines that are equivalent in that they encompass in toto exactly the same set of input-output functions. In effect, there is a single large frog pond of functions no matter what species of frogs (types of machines) is

used. But the frog pond is just a pond; it is not the whole ocean of all possible functions.

That there exists a most general formulation of machine and that it leads to a unique set of input-output functions has come to be called *Church's thesis*, after Alonzo Church, the logician who first put forth this claim with respect to one specific formulation (recursive functions) (Church, 1936). Church's statement is called a *thesis*, because it is not susceptible to formal proof, only to the accumulation of evidence. For the claim is about ways to formalize something about the real world, ie, the notion of machine or determinate physical mechanism. Self evidently, formal proof applies only after formalization. The most striking evidence has been the existence of different maximal classes of machines, derived from quite different formulations of the notion of machine or procedure, each of which turns out to be capable of producing exactly this same maximal set of functions.

A large zoo of different formulations of maximal classes of machines is known by now -- Turing machines, recursive functions, Post canonical systems, Markov algorithms, all varieties of general purpose digital computers, most programming languages (viewed as specifications for a machine). As a single neutral name, these classes are interchangebly called the *effectively computable procedures* and the functions that can be attained by the machines are called the *computable* functions.

These maximal classes contain universal machines, ie, machines that, if properly instructed through part of their input, can behave like any other machine in the maximal class. But then they can produce all the input-output functions that can be produced by any machine, however defined (ie, in any other maximal class). It is these machines that that are usually referred to as universal machines. From now on this is what we shall mean by *universal*. The proofs of the existence of these universal machines are also part of this early great chapter in the theory of logic and computers.

SS, our paradigmatic symbol system, is universal. Thus, it has as much flexibility as it is possible to obtain. It is useful to show that SS is universal. It is easy to do and its demonstration will make the notion transparent and keep it from accruing any mystery. Having the demonstration will also provide us with an example of a program in SS, which will clarify any confusing points in its definition. We will also be able to use this demonstration to support several points about general symbol systems, when we examine them in the next section.

To show SS is universal all we need to show is that it can simulate any member of a class of machines already known to be a maximal class. Let us choose the class of Turing machines: It is simple, classical, and everyone knows that it is a formulation of a maximal class.

At the top, Figure 3-1 shows a classical one-tape Turing machine. There is a single unending tape, with each cell holding a 0 or a 1. There is a control, which has a single reading head at a given cell of

the tape. The control is in one of a finite set of states, Q1, Q2, ..., Qn. For the control to be in a particular state implies it will do the following things:

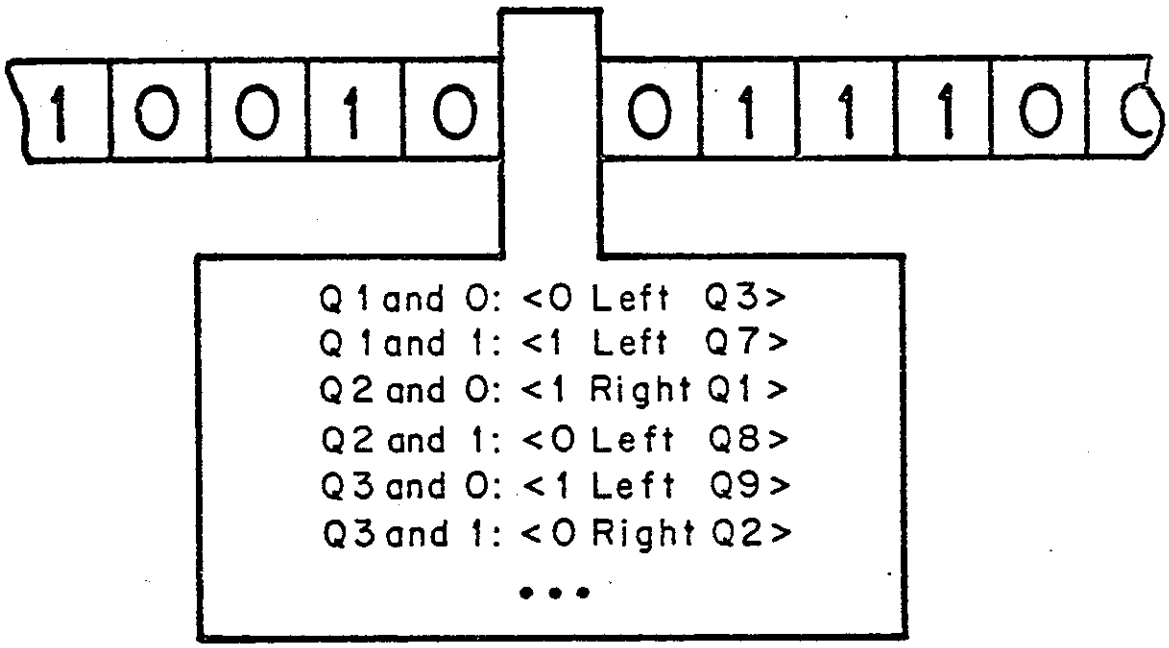
- It will read the symbol on the tape, ie, detect whether it is 0 or 1.
- It will write a symbol on the tape, either a 0 or 1 (as determined by the state and the symbol read).
- It will move the tape head one square, either to the left or the right (as determined by the state and the symbol read).
- It will change itself to be in a new state (as determined by the state and the symbol read).
- Instead of going to a new state, the machine may come to a halt.

Nothing is said about what sorts of physical arrangements are used to create the set of states for a given Turing machine and make it behave in the way specified. But we can write down for each state what it will do, and we have done this in Figure 3-1 for a couple of states (Q1, Q2, etc.). For each state we have two entries, depending on whether the tape cell has a 0 or a 1: the symbol to be written on the tape (either a 0 or a 1); whether to move left or right; and the state to go to. If a special next state, *halt*, is given, the machine halts.

Any machine built to behave this way is a Turing machine. Turing machines differ only in the number of states they have and in what happens at each state, within the limits described above. However, the class of all Turing machines is very large -- by using enough states (and it may take a very large number) the input-output behavior of any physical mechanism can be approximated as closely as required. It is one of these maximal class of machines, even though a Turing machine's moment by moment behavior seems very restricted.

The bottom of Figure 3-1 gives the program in SS that simulates an arbitrary Turing machine. The Turing machine itself must be represented. This is done entirely within the memory of SS, rather than in terms of the external input and ouput interfaces of SS. For any reasonable **input** and **behave** operators this extra translation would be straightforward. The representation uses three types of symbol structures, one for the tape cell and the other two for the state of the Turing machine control. The tape cell has three roles: *content* holds the tape-symbol, either 0 or 1; *left* holds the symbol for the tape cell to the left; and *right* holds the symbol for the tape cell to the right. The Turing machine state has two roles: *if-0* to hold the specifications for what to do if the tape cell holds 0; and *if-1* for what to do if the tape holds 1. Each specification (the third symbol structure type) has three roles: *content* holds the tape-symbol to be written, either 0 or 1; *move* holds the direction to move the tape, either *left* or *right*; and *next* holds the symbol for the next state to go to.

There is a single program expression, called TM, which accomplishes the simulation. TM consists




```
T3: [content:0  left:T2  right:T4]

Q2: [if-0:[content:1  move:right  next:Q1]
     if-1:[content:0  move:left   next:Q8] ]

TM: [do

        [do [read content T] [continue-if 0] [assign A [read if-0 S]]]

        [do [read content T] [continue-if 1] [assign A [read if-1 S]]]

        [write T content [read content A]]

        [assign T [read [read move A] T]]

        [assign S [read next A]]

        TM]
```

Figure 3-1: Simulation of arbitrary Turing machine by SS.

of **doing** a sequence of six subprograms, each of which accomplishes one step of the basic definition. There is a symbol T for the current tape cell under the head, a symbol S for the current state of the Turing machine control, and a symbol A, for the actions specified by the state (which will be different depending on the tape symbol). The program is shown as a single nested expression with many subexpressions. This is simply for convenience of reading; it does not indicate any additional properties of SS. Complex subexpressions are constructed entirely through the use of assignment. In each case what occurs in an expression is a symbol, which is assigned to the subexpression. Thus, the actual representation of TM is:

TM: [**do** TM1 TM2 TM3 TM4 TM5 TM]

TM1: [**do** TM11 TM12 TM13]

TM11: [**read** content T]

TM12: [**continue-if** 0]

TM13: [**assign** A TM131]

TM131: [**read** if-0 S]

TM2: ...

And so on through the whole expression.

Let us take up the steps of TM in order.

1. The first step reads the symbol on the tape and if it is 0 assigns the symbol A to be the actions specified in case 0 occurs, ie, the substructure at if-0.

2. The second step similarly assigns A to be the actions specified at if-1, if 1 is the tape symbol. (Only one of the two possible assignments to A will occur, since they are mutually exclusive.)

3. The third step writes the symbol specified via A into the tape-cell. This is a simple transfer of a symbol; it even occupies the same role in both structures, ie, content.

4. The fourth step moves the tape left or right by using the symbol specified via A as the role symbol to extract the left or right link from the tape-cell.

5. The fifth step is to assign S to be the next state as specified via A.

6. The sixth and final step is to do TM again, which repeats the entire interpretation, now on the changed values of T and S.

This demonstration of universality is transparent, because the universality of some given system (here, Turing machines) has already been established, and the system itself (here, SS) has reasonable

properties as a programming system.

Of course, it is necessary that the program be correct. In fact, two bugs exist in the present version. One arises because we didn't take care of halting. This requires more conventions: The Turing machine halts if the symbol *halt* occurs for the next state; then TM should exit and return to whatever program executed it, with the current tape cell (T) as output. The other bug arises because the tape for a Turing machine is of indefinite extent, while the representation of the tape in SS necessarily consists of only a finite number of symbol structures, one for each cell. It is necessary for the simulation to extend the tape in either direction, if it runs off an end. Again, by convention, the symbol *tape-end* occurring in a tape cell at either *left* or *right* will indicate the end of the tape.

Just for completeness, a correct version of the program appears in Figure 3-2. To correct the first bug, a top level program TM-Exec is defined, which simply executes TM and, when it is done, outputs T. Correspondingly, TM now tests if the new S is halt and exits if it is. For the other bug, TM senses whether the next tape cell is the symbol tape-end and, if so, extends the tape. This occurs en passant within the expression for moving the tape, [**assign** T [**read** [**read** move A] T]], by performing the regular operation within a **do**-sequence where it can continue if tape-end is found. It then creates a new tape cell (calling it New-T) and links it up in the several ways necessary. It ends by handing the new cell (as New-T) to the **assign** operator, just as if that cell had been found initially.

```
TM-Exec: [do TM T]
TM: [do
        [do [read content T] [continue-if 0] [assign A [read if-0 S]]]
        [do [read content T] [continue-if 1] [assign A [read if-1 S]]]
        [write T content [read content A]]
        [assign T [do [read [read move A] T]
                      [continue-if tape-end]
                      [assign New-T [copy T]]
                      [write New-T content 0 [read [read move A] other] T]
                      [write T [read move A] New-T]
                      New-T]]
        [assign S [read next A]]
        [exit-if halt]
        TM]
other: [right:left left:right]
```

Figure 3-2: Correct simulation of arbitrary Turing machine by SS.

4. GENERAL SYMBOL SYSTEMS

We can now describe the essential nature of a (physical) symbol system. We start with a definition:

Symbol systems are the same as *universal machines*.

It may seem strange to *define* symbol systems to be universal machines. One would think that symbol systems should be defined to be that class of systems that *has* symbols according to some abstract characterization. Then it would be a fundamental theoretical result that symbol systems were universal. However this way is not open to us, without a certain amount of scientific legerdemain. The fact is we do not have an independent notion of a symbol system that is precise enough to counterpoise to a universal machine, and thus subsequently to prove their equivalence. Instead, we have *discovered* that universal machines always contain within them a particular notion of symbols and symbolic behavior, and that this notion provides us for the first time with an adequate abstract characterization of what a symbol system should be. Thus, tautalogically, this notion of symbol system, which we have here called *physical symbol system*, is universal.

Does not SS, the machine we have just defined, provide a paradigmatic example that could be suitably generalized to define the class of symbol systems? True, SS was put together precisely to bring out the essential properties of symbols. Alas (for such an enterprise), SS and all its kindred have emerged simply as *reformulations* of the concept of universal machines. Historically, we are genuinely in the position of discovers, not inventers. For analytic purposes we can certaintly now propose axiomatic formulations of symbol systems and prove their equivalence to universal machines. But I prefer an exposition that emphasizes the dependence, rather than the independence, of the notion of (physical) symbol system on the notion of universal machines.

Thus, our situation is one of defining a symbol system to be a universal machine, and then taking as a hypothesis that this notion of symbol system will prove adequate to all of the symbolic activity this physical universe of ours can exhibit, and in particular all the symbolic activities of the human mind. In regard to our list of constraints of mind in Figure 1-1, two seemingly separate constraints (universality and using symbols) have been satisfied by a single class of systems.

We can now proceed to the essential nature of symbols and symbolic behavior in universal systems, and to their generality. Note, however, that universal machines provide a peculiar situation with respect to what is essential. Every universal machine exhibits in some form all the properties of any universal machine. To be sure, differences exist between universal machines -- in primitive structure, in processing times, in sensitivities, and in processing reliabilities. Though important -- even critical -- for some aspects of the phenomena of mind, these differences are not critical for the nature of symbols. Thus, when we focus on certain properties, we are providing an emphasis, rather than separating what cannot in truth be separated.

We start with a discussion of designation and interpretation. Then we go through the operators of SS. Though defined as specific operators for a specific machine, each corresponds to a general functional capability. Each operator thus raises the question of the necessity of this functional capability to a symbol system and also of the forms that it can take in alternative implementations while still accomplishing the essential function.

4.1. Designation.

The most fundamental concept for a symbol system is that which gives symbols their symbolic character, ie, which lets them stand for some entity. We call this concept *designation*, though we might have used any of several other terms, eg, *reference*, *denotation*, *naming*, *standing for*, *aboutness*, or even *symbolization* or *meaning*. The variations in these terms, in either their common or philosophic usage, is not critical for us. Our concept is wholly defined within the structure of a symbol system. This one notion (in the context of the rest of a symbol system) must ultimately do service for the full range of symbolic functioning.

Let us have a definition:

> *Designation*: An entity X designates an entity Y relative to a process P, if, when P takes X as input, its behavior depends on Y.

There are two keys to this definition. First, the concept is grounded in the behavior of a process. Thus, the implications of designation will depend on the nature of this process. Second, there is action at a distance: the process behaves as if inputs remote from those it in fact has effect it. This is the symbolic aspect, that having X (the symbol) is tantamount to having Y (the thing designated).

The symbols in SS satisfy this definition of designation. There are a set of processes (the operators and the control) to which symbols can be input, and when so input the processes behave as a function, not of the symbols themselves, but of what the symbols have been assigned to, what, therefore, they designate.

The question of what symbolization implies in SS can only be worked out by understanding the nature of these processes, which can now be called *symbolic processes*. That these processes taken together are sufficient for attaining universality states the biggest implication. That this universality is attained only because of the existence of symbols provides the conceptual knot that makes the notion deep.

In SS, the second aspect of the definition is provided by the mechanism of *access*, which is part of the primitive structure of SS. It provides remote connections of specific character, as spelled out in describing **assign**. This specification is generated by enumerating for each of the ten operators plus the control the precise access needed to carry out their specified operations. Exactly these and no

other forms of access are needed. This access is needed to exactly three types of entities: symbol structures, operators and roles in symbol structures. Thus, *access* is no homunculus, providing that this finite set of primitive properties can be realized in physical mechanisms. We already know, through our experience with digital computers, that this is not only possible but eminently practical.

The great magic comes because this limited capability for accessing supports a general capability for designation. The set of processes must be expanded to include programs, and their inputs must be taken to include expressions. Then, for any entity (whether in the external world or in the memory), if an expression can be created at time T that is dependent on the entity in some way, processes can exist in the symbol system that, at some later time T', take that expression as input and, behaving according to the recorded structure, behave in a way dependent on the entity. Hence these expressions designate the entity.

An important transitive law is illustrated in this, in which if X designates Y and Y designates Z, then X designates Z. In the case in point, there is first the acquisition which, through access to the actual external structure, creates a structure in the memory of the system that depends on this external entity; then the preservation of that memory structure through time yields a memory structure at some later time that still depends on the object; finally, the access associated with the internal symbol makes that structure available to a process, which then behaves accordingly, impressing it on still another entity and instantiating the relation of designation.

Because of the universality of symbol systems, the scope of this capability for designation is wide open and hardly yet explored. To repeat an earlier remark, the power of a designatory capability depends entirely on the symbolic processes to which it is coupled. If these processes are restricted enough, the total system may be able to accomplish little; if they are universal, then the total system may be able to do all that is required in human symbolization.

This general symbolic capability that extends out into the external world depends on the capability for acquiring expressions in the memory that record features of the external world. This in turn depends on the **input** and **behave** operators, whose details have not been described, but which limit access to the external world in some fashion. Such limits do not affect the capability of the symbol system to designate arbitrary entities, though they might limit the extent to which such capabilities could be utilized by a given system.

Designation is at the heart of universality. For one machine to behave as an arbitrary other machine, it must have symbols that designate that other. Once the input of the to-be-universal machine is separated into two parts, one of which is an instruction about something outside the machine (to wit, the other machine), there is no way out from generating some symbolic capability. That this symbolic capability should be general enough to encompass all notions of symbolic action

derives (if indeed it is true) from the scope of what was to be symbolized, namely any input-output function. But the kernel of the notion of symbols arrived by the single act of getting a machine to act like something other than it is.

A distinctive feature of SS is taking the general capability for symbols and access as central. Most formalizations of the notion of universal machine (all but those such as Lisp that stem from the work in artificial intelligence) take as central a more primitive capability for accessing, reflecting an interest in showing how universality can be built up by a machine. For instance, the Turing machine has symbols for the states of the control. These have the property of access, but are fixed and unchangeable -- symbols cannot be created or re-assigned. They do not provide the indefinitely extendable symbol system that is required for universality, but only some of the machinery for it. The indefinitely extendable symbol system is constructed as an addressing scheme on the (indefinitely extendable) tape. The construction is made possible by the tape movement operators, which provide the primitive accessing capability.

The underlying physical mechanism for obtaining access is some sort of switching mechanism, that opens up a path between the process and the thing accessed. There are a wide variety of such switching mechanisms, but they are closely related to *search*. If the medium is recalcitrant, eg, the Turing machine tape, the symbol system is implemented through a linear search of the tape and a match of a finite set of tape-symbols that serves to address the expression accessed. In more perspicuous media, eg, a preorganized addressing switch for a random access memory, the implementation takes features from the symbol token (the address) and uses them to construct a direct path to the requisite location.

The usual formulations of universal machines also tend to use the term symbol for the alphabet of distinctive patterns that can occur in the memory medium (eg, the 0 and 1 tape symbols for our Turing machine). As defined, these entities are not symbols in the sense of our symbol system. They satisfy only part of the requirements for a symbol, namely that of being the tokens in expressions. It is of course possible to give them full symbolic character, by programming an accessing mechanism that gets from them to some data structure. Table-look up mechanisms provide one scheme. Actually, the alphabet of such symbols is usually quite small (eg, only two for our version of a Turing machine), so they operate more like letters, ie, the elements out of which a genuine set of symbols can be constructed.

4.2. Interpretation

The term *interpretation* is taken here in the narrow sense it has acquired in computer science:

Interpretation: The act of accepting as input an expression that designates a process and then performing that process.

All of the behavioral flexibility of universal machines comes from their ability to create expressions for their own behavior and then produce that behavior. Interpretation is the necessary basic mechanism to make this possible. The general designatory capabilities of symbol systems underly the ability to create the designating expressions in the first place. Although little can be said about exact boundaries, some *interior milieu* must exist within which the symbol system can freely and successfully interpret expressions. Given this, obtaining other performances according to specification can be compromised in various ways, eg, by error, by indirect and shared control or whatever.

The symbols that designate operators are absolutely essential and no quantity of symbols for expressions or roles can substitute for them. These are the symbols that have an external semantics wired into them -- which finally solve Tolman's problem of how his rats, lost in thought in their cognitive maps, could ever behave. The number of such symbols can be shrunk by various encodings and parametrizations, but it cannot vanish. Equally (and more likely for real systems), the number can be much larger and the decomposition can be radically different than for SS.

The control exhibits a basic tripartite decomposition of the total processes of the machine, which can be indicated by *(control + (operators + data))*. Behavior is composed by one part, the control, continually bringing together two other parts, the operators and the data, to produces a sequence of behavior increments (ie, the operation formed by the application of the operator to the data). This will be recognized as familiar from every computer, programming language and mathematical system.[3] This structure can be taken as an essential organizational feature of all symbolic systems.

This organization implies a requirement for *working memory* in the control to hold the symbols for the operator and data as they are selected and brought together. Our description of SS in Figure 2-1 shows only the place for the active symbol and for the input and output symbols for the operators. This is the tip of the iceberg; perusal of Figure 2-3 shows that additional working memory is needed. What memory will vary with the type of universal machine, but some is always implied by the act of decomposition. Thus working memory is an invariant feature of symbol systems.

However, many things about the control of SS are not invariant at all over different types of universal symbol systems. One must be careful to treat SS as a frame for describing *functional* capabilities, abstracting away from many of its structural features. Three examples of this are:

[3]Mathematics exhibits the application of operator to data, ie, function to argument, while leaving the control indeterminate, ie, in the hands of the mathematician.

- SS's control requires an unbounded amount of memory (essentially a pushdown stack) because the nesting of programs can be indefinitely extended. This is inessential, though it makes for simplicity. Normally control is a fixed machine with fixed memory; and regular memory (which is unbounded) is used according to some memory management strategy to handle excessive embedding.

- SS has a serial control, ie, a single control stream. This is inessential, though it also makes for simplicity. There may be multiple control streams of all sorts. The **input** and **behave** operators may both be evoked and operate in parallel. There may be multiple controls, a few or many, functionally specialized or all equivalent, and interconnected in a variety of ways. The parallel units may themselves be universal machines, or limited controllers, or only operators. Under some conditions the resulting processing aggregate is not universal, under many it is.

- SS is a totally reliable system, nothing in its organization reflecting that its operators, control or memory could be errorful. This is inessential, though it again makes for simplicity. As was noted earlier, universality is always relative to physical limits, of which reliability is one. Once fundamental components are taken as having probabilities of failure, then ultimate performance is necessarily probabilistic. If failure probabilities are significant, the system organization can include counteracting features, such as checking, redundant processing and redundant memory codes. Up to a point universality can be maintained as a practical matter; at some point it is lost.

All of these complexities are important in themselves, and some of them lie behind other constraints in Figure 1-1. They do not seem of the essence in understanding the nature of symbolic capability.

4.3. Assign: The creation of designations

The function of the **assign** operator is to create a relation of access, hence of designation, between a symbol and an entity. It is, of course, limited to the access relations supported by the underlying machinery of SS: between SS's symbols and SS's expressions, roles and operators.

Assign implies several important properties:

At any time, a symbol designates a single entity.
Many symbols can designate the same entity.
A symbol may be used to designate any entity.

SS provides absolute and uniform adherence to these properties, but this is not necessary. For instance, from SS's simulation of a Turing machine, it can be seen that the requirements for multiple assignment and for reassignment of a symbol to an arbitrary entity are needed only for the small set of working symbols used in the TM program (T, S and A). All the other symbols (content, **do**, TM,...) can have fixed assignments. From this, the more general capability can be built up -- which is what programming the simulation demonstrates.

The situation here generally applies. Small amounts of the requisite capabilities can be parlayed into the full fledged capability. The minimal conditions are rarely interesting from a theoretical view, though successful elimination of an entire functional capability can be revealing. Minimal basic capabilities often imply gross inefficiencies and unreliabilities. Typical is the additional level of interpretation if simulation is used to recover the additional capabilities (as in our example). Thus, symbol systems that satisfy additional constraints of Figure 1-1 are likely to satisfy pervasively such properties as those above.

It is often observed that the symbols of formal systems are totally abstract, whereas symbols as used by humans often have information encoded into the symbol itself, ie, that it is not arbitrary what a symbol is used for. The word for not being happy is "unhappy", in which some knowledge about what the word designates is available from an analysis of the word itself. In plane geometry small letters (a, b, ...) are sides of triangles and capital letters (A, B, ...) are their opposite angles. In general, the use (and usefulness) of *encoded names* has no bearing on the basic nature of symbol systems. Encoded names can be taken as to be abstract symbols with bound expressions that provide the information in the encoded name. Thus, one expression has been granted a preferred access status. Though the assignment of symbols to entities has been limited, this will have an effect only if no freely assignable symbols remain as part of the system.

4.4. Copy: The creation of new memory

By applying **copy** whenever needed, SS obtains both an unbounded supply of expressions (hence of memory) and of symbols. That **copy** creates a copy is unessential, though by doing so it accomplishes a sequence of **reads** and **writes**. The essential aspect is obtaining the new expression and the new symbol. Neither can be dispensed with.

One of the few necessary conditions known for universal machines is:

A universal machine must have an unbounded memory.

The classical machine hierarchy of finite state machines, pushdown automata, linear bounded automata and Turing machines, expresses the gradation of capability with limitations in memory (Hopcroft & Ullman, 1969). Though essential, the condition of unboundedness is of little import, since what counts is the structure of the system. In all cases, the structure of the unbounded memory must eventually become uniform. Eg, ultimately SS has just a supply of undifferentiated *stuff* out of which to build expressions; the Turing machine has just a supply of undifferentiated tape cells. Thus, for every machine with unbounded memory, there are machines with identical structure, but bounded memory, that behave in an identical fashion on all environments (or problems) below a certain size or complexity.

The unimportance of actual unboundedness should not be taken to imply the unimportance of large memory. The experience in AI is everlastingly for larger effective memories (ie, memories with adequately rapid access). A key element in list processing was the creation of dynamic memory, which effectively removed the memory limit problem from the operation of the system, while, of course, not removing it absolutely (ie, available space eventually runs out). It is no accident that humans appear to have unbounded long term memory. Thus, rather than talk about memory being actually unbounded, we will talk about it being *open*, which is to say, available up to some point, which then bounds the system's performance, both qualitatively and quantitatively. *Limited*, in opposition to open, will imply that the limit is not only finite, but small enough to force concern. Correspondingly, *universal* can be taken to require only sufficiently open memory, not unbounded memory.

Symbols themselves are not memory; only expressions are. Though in SS symbols and expressions come into existence together, they are independent and could have separate **create** operators. Many symbols may be assigned to a single expression and many expressions may have the same symbol (over time) or may be unsymbolized and be accessible through other means. Symbols are the patterns in the symbol structure that permit accessing mechanisms to operate. Having an open number of symbols, but only a limited amount of memory, isn't sufficient for a universal machine. On the other hand, with only a limited set of symbols, but an open supply of expressions, it is possible to create an open set of symbols. The use of a limited alphabet to create words is paradigmatic. However, just the anatomy of alphabets and words does not reveal the key issue, which is the construction of an accessing mechanism that makes the words behave like symbols, ie, designate.

SS actually has an open supply of expressions of each type (and exactly what types exist was not specified). As might be expected, only a single source of openness is needed, providing it is not peculiarly tucked away, as in a pushdown stack. Further, SS's definition does not specify whether expressions themselves are limited or whether some of them can be open. This is again an unessential issue, as long as at least one open source is available for construction of whatever facilities are needed. The creation of an open structure-type, the *list*, out of an open set of expressions of a limited structure-type, the *pair* consisting of a *symbol* and a *link*, is paradigmatic. Though conceptually simple, such a construction was a major step in creating appropriate symbol systems.

4.5. Write: The creation of arbitrary expressions

Another obvious, but important, necessary capability of a universal machine is:

A universal machine must be able to create expressions of arbitrary character.

SS does this through a single uniform operator, **write**; though there are indefinitely many complex and indirect ways of attaining the result. To be unable to create an expression, by any means at all, would imply a failure to be universal (eg, to simulate a machine that did produce that expressions as an output).

In the usual way of specifying universal machines, particular representations are used for the expressions, eg, the Turing tape or Lisp lists. Much of the idiosyncracy of such systems arises from the need to encode all structures of interest into this fixed structure. SS has remained general on this score, admitting only a basic capability for having expressions with distinct roles. Thus, we simply defined a new data type for each entity we needed to discuss, eg, programs, tape cells, and machine states.

It is unclear how to state the fundamental capability provided by expressions, though easy enough to exhibit it in simple and paradigmatic form. It is not enough to have only symbols. Expressions permit more than one symbol to be brought together in a way that is not determined wholly by the symbols, but provides additional structure, hence discriminability. This is what SS has in the roles -- actually, in the association from the role symbol to its content symbol. There is an indefinite number of ways to provide such added structure to yield symbol expressions.

In SS's scheme, the symbols for roles are *relative* symbols. They differ in this respect from the symbols for expressions or operators, which are *absolute*. Given the symbol *move*, which designates the role in the tape-cell of the Turing machine, the processes that take roles as input, namely **write** and **read**, can access the appropriate location in any tape-cell expression. Thus, role symbols are functions of one input (the expression), and akin to operators. These relative role symbols can be replaced by absolute symbols that uniquely designate the locations in particular expressions, though an additional operator is required to obtain these location symbols. This is the standard recourse in common programming languages, which provide *addresses* of list cells and array cells. Thus, all symbols can be absolute, with all context dependence relegated to a limited set of operators.

4.6. Read: Obtaining the symbols in expressions

Read is the companion process to **write**, each being necessary to make the other useful. **Read** only obtains what was put into expressions by **write** at an earlier time; and a **write** operation whose result is never **read** subsequently might as well not have happened.[4]

[4]This is too strong: the read operator is not the only process that reads expressions, control at least reads programs; and if the expression might have been read but wasn't because of a contingency in the environment, then the write operator still would have been useful, analogous to insurance that is never cashed.

The **read-write** coupling emphasises another necessary principle of symbol systems:

Memory must be stable.

Though much less investigated than the question of amount of memory, this principle is of the same character. In so far as memory is unreliable, the ability of the symbol system to deliver a given input-output function is jeopardized. Such a limitation does not destroy the functional character of a symbol system; it only modulates it. Of course, different systems behave differently under unreliability, and systems can be designed to mitigate the effects of unreliability. Such considerations are outside the bounds of the paper (though they show up as one of the constraints in Figure 1-1.)

In SS the reading operator was defined in the classical way, namely, a local operator whose scope was a given expression. This provides no global access to the memory. Indeed, SS is totally dependent on the initial data structures to provide linkages around the memory. A more global accessing operator could also be given:

Find the expression that matches roles (**find** R_1 S_1 ...)
Produces the expression or nil

In SS, attention must be paid to constructing access links to the new expressions created by **copy**; this is usually done by virtue of their symbols occurring in other expressions that are being built. Given other processing organizations, such as ones with parallel activity, then a **find** operation would be necessary, since the access links could not exist for **read** to suffice as a retrieval mechanism.

Such a global operator cannot as it stands replace the local one, since it identifies expressions by content, not by role in some other expression. However, versions can be created to combine both functions.

4.7. Do: The integration and composition of action

To be able to describe behavior in expressions, the behavior must be decomposed, whether for a description indirectly as a machine that generates the behavior, or any other type of description. All such decompositions involve both primitives and combining schemes. For SS, doing a sequence of operations is the combining operation. It reflects directly a necessary requirement on decomposition:

Universal machines must be able to determine the future independent of the past.

Looked at in terms of a pre-specified input-output function, it is only necessary that the future beyond the point of instruction be open. But if new instructional input is permitted at any time (the realistic version of the flexibility constraint), then any commitment for the entire future can be a genuine restriction of flexibility. Instruction following such a commitment might require its undoing.

Other forms of decomposition exist besides the pure *time-slice* scheme used by SS (and also by

existing digital computers and programming languages), in which each operator specifies completely what happens in the next time increment, leaving complete freedom of specification beyond. For instance, the commitments of the past could decay in some way, rather than cease abruptly; the future action, while still free to be anything, could be taken from a different base line in some way. Little is known about such alternative decompositions in terms of providing universal instructable behavior.

Do as an operator can be eliminated, because its essential function is also performed by the control. Interpreting the arguments of an operator prior to executing that operator (which corresponds to function composition) provides the essentials of a time decomposition. Thus, the function would still be provided, even though the **do** operator itself were eliminated.

4.8. Exit-if and Continue-if: The conversion of symbols to behavior

One of the most characteristic features of programming languages is the existence of conditional operators, the **if-then-else** of Algol-like languages, the **branch on zero** of machine languages, or the **conditional expression** of Lisp. These operators seem to contain the essence of making a decision. Beside embodying the notion of data dependence in pure form, they also are unique in embodying the conversion from symbols to behavior. They would appear to be a functional requirement for universality. They do not seem so much deeply implicated in the concept of symbols itself, but rather associated with the system that operates with symbols.

However, it has been known since the earliest formulations of universal machines that such conditionals are not uniquely required. The requirement is to be able to compose all functions, and many other primitive functions provide the essential combinative services. For example, the minimum function is often used. It can be seen that taking the minimum of a set of elements effects a selection, thus making a decision. But the minimum function has no special symbol-to-behavior character; it has the same form as any other function.

Thus, conditionals are a convenience in SS. They can be dispensed with and the same work done by **assign**, **copy**, **write**, **read** and **do**. A simple way to show this is to write the simulation of the Turing Machines without using the conditionals. The universality of SS is shown by this program; hence the universality can be attained by whatever limited means are sufficient to accomplish this simulation. Figure 4-1 shows this simulation without the conditional. This corresponds to the completely correct simulation of Figure 3-2, so that all places where conditionals originally occurred are covered.

It is instructive to understand the basic device in Figure 4-1. Conditionality is the dependence of behavior on data. The data in the simulation are symbols. Hence, each possible data symbol can be

```
TM-Exec:  [do [assign [quote Next-step] [quote TM]] TM T]

TM: [do
        [assign 0 [read if-0 S]]
        [assign 1 [read if-1 S]]
        [assign A [read content T]]
        [write T content [read content A]]
        [assign T [read [read move A] T]]
        [assign S [read next A]]
        Next-step]

tape-end: [do
        [assign New-T [copy T]]
        [write New-T content 0 [read [read move A] other] T]
        [write T [read move A] New-T]
        New-T]

other: [right:left left:right]

halt:  [assign [quote Next-step] nil]
```

Figure 4-1: Elimination of conditional operators from simulation of Turing machine.

assigned whatever is to be the case if that data symbol is encountered. The symbol that actually occurs and is accessed brings with it the behavior to be taken.

There are three places where conditionals must be removed. The first is taking appropriate action, depending on 0 or 1. For this, 0 is assigned the action specifications for 0, and 1 the action specifications for 1. Then accessing the symbol that actually occurs in the tape cell, via [**read** content T], obtains the action specification as a function of the occurring state. This is assigned to the termporary working symbol, A, and the program can proceed as before.

The second place is sensing the end of the tape. Here, the symbol tape-end is assigned to be the program that extends the tape. Due to the recursive interpretation of control, accessing tape-end leads to the interpretation of this program, which then fixes up the tape en passant. Thus, the main program, TM, never has to deal with the problems explicitly.

The third place is exiting on halt. This is a little tricky, because a symbol (halt) must be converted to a permanent change of behavior (exiting the infinite loop of TM). The final step of TM, which is the recursive step to repeat TM, is made into a variable symbol, Next-step. This is assigned to be TM in TM-Exec, so that if nothing changes this symbol, TM will be repeated, just as before. The symbol halt is assigned to a program that assigns Next-step to be nil. Thus, if halt is encountered, this program is executed, making TM into a straight-line program, which will then exit. It is necessary to use **quote** in the **assignments** of program symbols (Next-step and TM), or they would be executed inadvertantly.

This style of programming illustrates an important relativity of view. From the perspective of the program there is no choice and no decision; it simply puts one foot in front of the other so to speak. From the perspective of the outside observer a choice is being made dependent on the data. The views are reconciled when it is seen that the program is constructing its own path, laying down each next step just in front of itself, like a stepping-stone, and then stepping onto it.

4.9. Quote: The treatment of processes as data

All universal systems contain a distinction between operators and data (ie, arguments). To create its own procedures, a system must distinguish some expressions as data at one time (when creating or modifying them) and as program at another time (when interpreting them). This is a genuine contextual effect, since the expression is to be same in either case. This is only a binary distinction, and it can be achieved in many ways: by having a program memory as distinct from a data memory, with a transfer operation to activate the program; by marking the program so the control won't interpret it and then removing the mark to activate it; by having an **execute** operator that only interprets under deliberate command; or by having a **quote** operator that inhibits interpretation on command. For SS, since its control cycle is to interpret everything, the **quote** command is the natural

choice.

However, as Figure 3-1 again shows by producing the simulation without the use of **quote**, this operator does not imply an additional primitive functional requirement. What Figure 3-1 actually shows is that if a symbol system is willing to operate in indirect simulation mode, the distinctions can be introduced by data conventions. This is because the control of the system now becomes programmable.[5]

4.10. Behave and Input: The interaction with the external world

Behave and **input** have played a muted role in the exposition of SS only because the emphasis has been on the basic functional requirements for symbolization and for universality. These capabilities must exist in some interior system, and thus can be illustrated there, without involving the interaction with the external world.

Behave and **input** imply an extension of the basic access mechanism, beyond the operators, roles and expression, as described above.. The symbols that are operands in **behave** must access in some way the effector mechanisms. These symbols can be viewed simply as additional operators which haven't been specified because there was no need to. **Input**, on the other hand, requires its output symbols to reflect an invariant relation to the state of the external environment (via states of the receptor mechanism). The invariance doesn't have to be perfect; it can even change over time (though not too rapidly). But without some reliable transduction from external structure to symbols, the symbol system will not be able to produce reliable functional dependence on the external environment.

General symbol systems include unlimited elaborations on **behave** and **input**. In particular, versions of these operators need not link to a genuine external world, but simply to other components of the total system that provide additional computational devices. These may be integral to the actual operation of the system in practice. Such additional facilities do not destroy the capability for symbolic action. The only requirement is that they be symbolizable, ie, have symbols that evoke them and symbols that reflect their behavior.

4.11. Symbol systems imply universality

The notion of universality has been expressed so as to reveal how it contains a notion of symbol and symbol system. Though at the beginning of the section I claimed it was inappropriate to act as if

---

[5]The quotes can likewise be removed from Figure 4-1 by rewriting the TM program so it is a non-program data structure that is interpreted by an SS program.

we had an independent characterization of symbol system, it is now certainly useful to extract from the current formulation what a symbol system might be, independent of the notion of universality. For instance, this might lead to interesting variants of symbol systems that are not universal in some important way (ie, in some way other than physical limits).

Consider the following generalized characterization of the notions involved in physical symbol systems:

- Symbols as abstract types that express the identity of multiple tokens.
- Expressions as structures containing symbol tokens.
- Designation as a relation between a symbol and the entities it symbolizes.
- Interpretation as realizing the designations of expressions.
- Operations of assigning symbols, and copying, reading and writing expressions.

These notions all seem fundamental to symbolic functioning. It is difficult to envision a notion of symbol that does not embody some version of these capabilities, though perhaps much more besides. These notions are too vague actually to define symbolic functioning. For instance, the statement about designation does not describe the properties of the relation, eg, between the word "cat" and the animal cat. The statement about interpretation leaves entirely open what symbolic activity is actually about -- it could easily hide a homunculus. Still, these might form the thematic kernel from which to precipitate independent characterizations of symbols.

In particular, there exists an instantiation of these notions that regains the formulation of a physical symbol system. The chief additional ingredient (abstracted away in generating the list above) is a notion of symbolic *processing*. Designation is given a primitive basis primarily in the accessing of other expressions. Interpretation is given a formulation involving only expressions that designate symbolic processing. Assigning, copying, reading and writing are taken as specific processing functions; in particular, reading is taken only as the ability to obtain constituent symbols. These particularities would no doubt be included in some fashion in any instantiation of the general notion of symbols stated above. However, the instantiation for physical symbol systems is still highly special and much is missing: designation of external entities, wider ranges of interpretive activity, and so on.

Yet, as we have seen, this process-oriented instantiation of these notions is by itself sufficient to produce universality. No embedding of the symbol system into a larger processing system with other capabilities is required, though sufficient freedom from physical limitations (ie, sufficient memory, reliability, etc.) must exist. In the preceding discussion, the operations of **exit-if**, **continue-if**, **do**, **quote** and **find** were shown to be collectively unnecessary to achieve universality. Thus, the

operations that appear inherently involved in symbolic processing (**assign**, **copy**, **read**, **write** and **interpret**) are collectively sufficient to produce universality. No augmentation with any non-symbolic processing machinery is required. Though the argument was carried through on a particular system, SS, it applies generally to the functional capabilities themselves.

A novel feature of physical symbol systems is the approach to symbolic function, not just by processing, but by *internal* symbolic processing. The primitive symbolic capabilities are defined on the symbolic processing system itself, not on any external processing or behaving system. The prototype symbolic relation is that of access from a symbol to an expression, not that of naming an external object. Thus, it is an implication of the formulation, not part of its definition, that the appropriate designatory relations can be obtained to external objects (via chains of designation). Because of this, the exact scope of that designatory capability is left open, implicit in the ramifications of universality.

Thus, we are lead finally to the following hypothesis:

Any reasonable symbol system is universal (relative to physical limitations).

It is important to distinguish symbol systems that are computationally limited because of physical constraints or limited programs and data, from symbol systems that fall short of providing universality because of structural limitations. The hypothesis refers to the latter.

Despite this hypothesis, one might still want to formulate a notion of symbol system that was not also universal, even though it would be limited. One general path might be to deny the process base. But this seems unfruitful, since symbol systems must ultimately be used by processing systems, and this path simply keeps the processing implications off stage. The addition of a processing base would very likely simply convey universality. Another possibility is to consider systems with many symbol systems, each ranging over severely limited domains, with limited intercommunication. These would violate some aspects of assignment, so that genuinely limited systems might emerge. But, in general, looking for a more limited conception of symbolic system, in order to get something conceptually independent of universality, does not seem particularly rewarding. This seems especially the case in trying to understand the human mind, which surely exhibits extreme flexibility, even though it must cope with some stringent physical limitations.

5. THE PHYSICAL SYMBOL SYSTEM HYPOTHESIS

Having finally made clear the nature of a physical symbol system, the major hypothesis can be stated explicitly (Newell & Simon, 1976):

- *Physical Symbol System Hypothesis*: The necessary and sufficient condition for a physical system to exhibit general intelligent action is that it be a physical symbol system.
- *Necessary* means that any physical system that exhibits general intelligence will be an instance of a physical symbol system.
- *Sufficient* means that any physical symbol system can be organized further to exhibit general intelligent action.
- *General intelligent action* means the same scope of intelligence seen in human action: that in real situations behavior appropriate to the ends of the system and adaptive to the demands of the environment can occur, within some physical limits.

The hypothesis takes as given the identity of symbol systems and universal systems, and asserts their connection to rationality, a concept which did not enter into their formulation. The hypothesis implicitly asserts that physical symbol systems cover human symbol systems, since general intelligence includes human intelligence. It can be taken as also asserting the essential role of human symbols in human rational behavior, if that cannot be taken for granted.

The hypothesis implies that symbol systems are the appropriate class within which to seek the phenomena of mind. However, it does not mention *mind* explicitly, but rather the notion of general intelligent action. It thereby implicitly takes general intelligence to be the key to the phenomena of mind. Given the democracy of the constraints in Figure 1-1, this may seem a little presumptuous. If so, it is not a presumption that makes a substantial difference in the short term. The systems that satisfy all of the constraints will undoubtedly be a highly distinctive subclass of those that satisfy only the three involved in the hypothesis -- universality, symbols, and rationality. This distinctiveness could well include phenomena of mind that would make the total class appear quite unmind-like. That possibility does not affect the tactical issue of approaching the phenomena of mind via this class of systems.

The statement of necessity is straightforward. A general intelligent system, whatever additional structures and processes it may have, will contain a physical symbol system. It will be possible to find what serves as symbols and as expressions; and to identify what processes provide the functions that we have enumerated and discussed. The variability of realization discussed in the prior section may make these structures and processes far from obvious, but they will exist.

The statement of sufficiency requires a little care. A universal system always contains the potential for being any other system, if so instructed. Thus, a universal system can become a generally

intelligent system. But it need not be one. Furthermore, instructability does not imply any ability at self instruction, so that there may be no way to transform such a system into one that is generally intelligent, ie, no external agent with the capability to successfully instruct it need be available. Given the nature of universality, this sufficient condition does not have much bite; it is the necessary condition which carries the strong implications.

The notion of general intelligence can only be informally circumscribed, since it refers to an empirical phenomenon. However, the intent is clear -- to cover whatever will come to be called intelligent action as our understanding of the phenomenon increases. The term *general* excludes systems that operate only in circumscribed domains. If the domain is narrow enough, considerable intellectual power may be possible from systems that are not physical symbol systems. Thus, a specific enumeration algorithm for chess that achieved master level, but was realized directly in hardware in a way that avoided the full capabilities of a symbol system, would not provide a counterexample to the hypothesis. General intelligence implies that within some broad limits anything can become a task. It would suffice to ask if the given narrow algorithm could also accept other novel tasks; and on this it would, per hypothesis, fail.

All real systems are limited. To be generally intelligent does not imply the ability to solve or even formulate all problems. We have used the phrase *physical limits* to indicate the effects of underlying limits to speed, memory size, reliability, sensitivity, etc. The existence of such limits implies the possibility of quibbles in assessing the hypothesis, if the limits are so stringent as to deny a system any reasonable scope for positive performance. The formulation above does not attempt to be precise enough to deal with such quibbles.

5.1. Why might the Hypothesis Hold?

That the hypothesis refers to rationality, rather than more generally to phenomena of mind, is not just a rhetorical preference. The hypothesis is based on the empirical evidence of the last twenty years in artificial intelligence. That evidence specifically relates to rational goal-directed behavior, and not to the other constraints (though some evidence exists touching one or two others). Thus, the hypothesis really must be cast in this narrower form.

It is important to understand that the hypothesis is empirical and rests on this body of experience. Artificial intelligence has made immense progress in developing machines that perceive, reason, solve problems, and do symbolic tasks. Furthermore, this has involved the deliberate use of symbol systems, as witnessed in the development and exploitation of list processing. This use of symbolic computation distinguishes artificial intelligence from most other enterprises within computer science (though not all). These advances far outstrip what has been accomplished by other attempts to build intelligent mechanisms, such as the work in building robots driven directly by circuits, the work in

neural nets, or the engineering attempts at pattern recognition using direct circuitry and analogue computation.[6] There is no space in this paper to review the evidence for this, which covers the development of an entire field over almost a quarter century. Reference to the most recent textbooks will have to suffice (Nilsson, 1980, Winston, 1977).

Given our present understanding of intelligent programs, an analysis can be made of why symbol systems play a necessary role in general intelligent action. Again, there is no space to do more than outline this analysis here. There seems to be three main points.

1. A general intelligent system must somehow embody aspects of what is to be attained prior to attainment of it, ie, it must have *goals*. Symbols that designate the situation to be attained (including that it is to be attained, under what conditions, etc.) appear to be the only candidate for doing this. It might seem an alternative to build goal-orientation into the structure of the system at design time (as is often done is programs that have a single fixed task, such as playing a game). However, this does not suffice for a general intelligence facing an indefinite sequence of novel and sufficiently diverse goal situations.

2. A general intelligent system must somehow consider candidate states of affairs (and partial states) for the solutions of these goals (leading to the familiar search trees). Symbols in a symbol system appear to be the only way to designate these, especially as the diversity and novelty of the states and partial states increases without bound.

3. An intelligent system must fashion its responses to the demands of the task environment. As the diversity of tasks expand, ie, as the intelligence becomes general, there would seem to be no way to avoid a flexibility sufficient to imply universality and hence symbols..

The backbone of the above argument is: (1) rationality demands designation of potential situations; (2) symbol systems provide it; (3) only symbol systems can provide it when sufficient novelty and diversity of task is permitted. This latter aspect is analogous to standard arguments in linguistics concerning the implications of generation of novel sentences.

---

[6]The term *direct* is used as a shorthand to indicate that the systems do not use digital computers as a major component.

6. REALIZATIONS AND SYSTEM LEVELS

Symbol systems, as described, are abstract. We now need to consider their realization in our physical universe. The key to this is our experience with the construction of digital computers. That current digital technology involves a hierarchy of levels is well known and appreciated. However, it is part of the story of symbol systems and needs to be recounted briefly.

A standard set of levels has emerged as digital computers have developed. These levels are levels of *description*, since it is always the same physical system that is being described. Each level consists of characteristic components that can be connected together in characteristic fashion to form systems that process a characteristic medium. The different descriptions form a sequence of levels, because the components, connections and media of one level are defined in terms of systems at the next lower level.

The bottom-most level starts with the description of the physical devices in electronic terms. It is usually called the *device* level. Above this is the *circuit* level, which consists of electrical currents and voltages, traveling in wires. Above that is the *logic* level, in which there occur registers containing bits, with transfer paths between them and various logical functions occurring when bits pass through functional units. Operation here is entirely parallel, as it is at all lower levels. The next level is the *program* level which contains data structures, symbols (or variables), addresses and programs. Operation is sequential, consisting of control streams produced by interpreters, though concurrent control streams may exist. This is the level of the symbol system as it occurs in digital computers. Above the programming level is the level of gross anatomy, the so-called *PMS* (*Processor-Memory-Switch*) level. Here there is simply a medium, called data or information, which flows along channels called links and switches and is held and processed by units called memories, processors, controls and transducers. It is the level at which you order a computer system from the manufacturer.

Each of these levels provides a complete description of a system, ie, one in which the present state of the machine plus the laws of behavior of the system (at that level) determine the entire trajectory of the system through time.[7]

Although apparently only a way of describing the physical world, each level in fact constitutes a *technology*. That is, any description of a system can be realized physically, because physical techniques exist for creating the required components and assembling them according to the description. Circuits of any description can be built, so also logic circuits, so also programs with any

[7] The top level (PMS) is often an exception, for behavioral laws are not usually formulated for it.

data types and routines. At the PMS level, computer configurations can be ordered with various combinations of memory boxes, disks and terminals. And so on. (Limits do exist to the realizablility of arbitrary descriptions, eg, the number of nested expressions in a programming language or the fanout in a logic circuit technology; these complicate, but do not destroy, the technological character of a level.) Thus, these levels of description do not exist just in the eye of the beholder, but have a reality in this combinative characteristic in the real world. The levels are not arbitrary and cannot be created at will, just by an act of analysis. On the other hand, there is no persuasive analysis yet that says this particular set of levels is necessary or unique, and could not be replaced by a quite different set.

From the prior discussion of symbol systems we should be prepared for the existence of an indefinitely wide variety of symbol systems. Such variety stems from all the different forms of operators, controls, memories and symbol-structures, that still add up to universal symbolic capability. The logic level structure that creates a particular symbol system is called the *architecture*. Thus, there are an indefinite variety of architectures. Indeed, they are so diverse that we have no reasonable characterizations of the class of all architectures.

What we had no right to expect is the immense variety of physical ways to realize any fixed symbol system. What the generations of digital technology have demonstrated is that an indefinitely wide array of physical phenomena can be used to develop a digital technology to produce a logical level of essentially identical character. If evidence must be quoted for this, it comes in the form of the *architecture family*, achieved first by IBM in the mid sixties with System/360 and now provided by most manufacturers, whereby many implementations exist for a given architecture, trading cost for speed and memory capacity. Programs that run on one implementation also run on the other. Furthermore, these implementations are not all planned for in advance, but as brand new technologies gradually come into existence at the device level, new implementations of the existing architecture are created.

Thus the picture that emerges is a series of levels of technology, with a many-many mapping between levels -- each level giving rise to an immense diversity of systems at the next higher level, and each system at a given level being realizable by an immense diversity of organizations at the next lower level.

That humans are physical symbol systems implies there exists a physical architecture that supports that symbol system. The relatively gross facts about the nervous system reveal some natural candidates for the levels of organization at which technologies exist. The neural level surely constitutes a technology. So also does the macromolecular level (in fact, several technologies may exist there). It is possible to be mistaken about the levels, given the potentiality at the macromolecular level, as seen, for instance, in the immune system. But such uncertainty does not destroy the

essential picture:

> There must exist a neural organization that is an architecture, ie, that supports a symbol structure.

Furthermore, the immense diversity of lower level technologies that can lead to an architecture certainly enhances the chance that a biologically based architecture could have evolved.

This is a genuine prediction on the structure of the nervous system and should ultimately inform the attempt to understand how the nervous system functions. It does not appear to have done so, though from time to time the suggestion has even been made directly (eg, Newell, 1962). In fact, I know of no discussion of the issue in the neuroscience literature.

The reasons for this lack of attention by the neurosciences lie beyond the present paper. Some of the considerations are evident in Geschwind's paper at this conference (Geschwind, 1979), where emphasis is placed on the special purpose computational systems that seem to be available in the organism, even to doubting that any general purpose mechanisms exist. As the present exposition should make clear, the requirement for universal symbolic functioning is not incompatible with extensive special purpose computational structure. It implies neither that everything must be done through programming a small set of primitive facilities nor that the symbol system occur as an isolated component. To take SS (or similar examples of formally defined universal systems) as implying such properties, fails to appreciate the actual functional requirements they express.

The levels structure of physical implementation, and our experience with it for digital technologies, leads to understanding how one level can be *sealed off* from its lower level during normal operation. This is the phenomenon of not being able to identify *under normal conditions* the technology in which a computer is implemented, if access is available only to the behavior at the symbolic level. This sealing off produces an effect in which the symbolic behavior (and essentially rational behavior) becomes relatively independent of the underlying technology. Applied to the human organism, this produces a physical basis for the apparent irrelevance of the neural level to intelligent behavior. The neural system is not in fact irrelevant -- its operation supports the symbolic level. But it does so in a way that normally hides most of its properties, realizing instead a symbol system with properties of its own.

The phrase *under normal conditions* is essential to the above characterization. Errors of all sorts that occur at lower levels typically propagate through to higher levels (here, the symbolic level) and produce behavior that is revealing of the underlying structures. Likewise, simply forcing a system against the physical limits of its behavior reveals details of the underlying technologies. Given a stop watch, the freedom to specify the tasks to be performed on a computer, and a system that is not designed to deceive, much can be learned of the lower levels of implementation. Similarly, if the

system uses large subsystems of special computational character, these too may reveal themselves.

This entire story of technological system levels, and the many-many relationship of systems on the symbol level to architectures that support it, is an important part of the current knowledge about symbol systems. Like the link to rational behavior (as expressed in the basic hypothesis), it is primarily empirically based.

7. DISCUSSION

With the basic story now before us, a few issues can be touched on to make sure that the notion of symbol system and the hypothesis are correctly understood.

7.1. Knowledge and Representation

Two terms intimately related to symbolic behavior have not appeared in the discussion so far: *representation* and *knowledge*. Both have rather clear meanings within the concept of physical symbol system, especially in the practice of artificial intelligence. However, formal theories of these concepts are relatively chaotic, with little agreement yet. Still, it is useful to indicate the sense of these notions, albeit briefly.

Representation is simply another term to refer to a structure that designates:

> X *represents* Y if X designates aspects of Y, ie, if there exist symbol processes that can take X as input and behave as if they had access to some aspects of Y.

The qualification to aspects of Y, rather than just Y, simply reflects language usage in which X can be said to represent a complex object Y without being faithful (ie, designating) all aspects of Y.

Representation is sometimes formulated in terms of a mapping from aspects of Y to aspects of X. Implicit in this formulation is that something can be done with X, ie, that processes exist that can detect the aspects of X that are images of aspects of Y. Hence the whole forms a designatory chain.

Representation is also sometimes formulated in terms of a data structure with its associated *proper* operations. This view emphasizes the coupling of the static structure (what is often simply called *the* representation) and the processing that defines what can be encoded into the structure, what can be retrieved from it and what transformations it can undergo with defined changes in what is represented. This view suppresses the function of the memory structure (ie, what it represents) in favor of the essential mechanics, but it comes to exactly the same thing as the formulation in terms of designation.

The term representation focusses attention on the image of the distal object in the symbolic structure which represents it. The analysis of universality and symbols, as presented here, focusses on the adequacy of constructing functions from the distal object to the behavior of the system, which work through the representations as an intermediate structure. Such a presentation leaves undeveloped the structure of descriptive schemes, with the corresponding questions of efficiency and usefulness. We saw a reason for this in the initial formulation of universality, where it was important to avoid confounding the limitations of descriptive schemes for possible functions with what functions could actually be produced by a machine.

Existing work, mostly stemming from the analysis of formal logic, confirms that the class of systems described here (ie, universal symbol systems) is also the class of systems with general powers of representation or (equivalently) description. The representational range of all first order predicate calculi is the same, and corresponds to universal systems (when one asks what functions can be described in the logic). An important chapter in logic was the demonstration that set theory, perhaps the most useful descriptive scheme developed in mathematics, was formulable in first order logic, thus becoming simply another alternative descriptive scheme, not one capable of describing a different range of entities and situations. Higher order logics (which progressively remove restrictions on the domains of variables in logical formula), do not extend the expressive range. Modal notions, such as *possibility* and *necessity*, long handled axiomatically in a way that made their relationship to standard logic (hence universal symbol systems) obscure, now appear to have an appropriate formulation within what is called *possible world semantics* (Hintikka, 1975, Kripke, 1972), which again brings them back within standard logic. The continuous functions that naturally occur in the world (hence, must be represented) are produced by systems of limited energy. Hence, they must be of limited frequency (ie, limited *bandwidth*) and have, by the so called *sampling theorem*, adequate finite discrete representations.

The above rapid transit through some basic theoretical results on representation is meant to indicate only two things: first, some general things are known about representation; and second, representation is intimately tied to symbol systems. Much more is known in empirical and practical ways about representation, especially from investigations of artificial intelligence systems. However, no adequate theory of representation exists for questions of efficiency, efficacy and design -- the level at which most interesting issues arise.

Knowledge is the other term that has not figured as prominently in our discussion as might have been expected. It is a competence-like notion whose nature can be indicated by the slogan formula:

Representation = Knowledge + Access.

Given a representation, making use of it requires processing to produce other symbolic expressions (or behavior). Although it is possible for a symbolic structure to yield only a small finite number of new expressions, in general there can be an unbounded number. Consider what can be obtained from a chess position, or from the axioms of group theory, or from a visual scene. Further, to obtain most of these new expressions requires varying amounts of processing. Thus, it is theoretically useful to separate analytically the set of potential expressions that a representation can yield from the process of extracting them, ie, the access to them. Knowledge is this abstract set of all possible derived expressions.

This notion, which corresponds to the set of all implications of a set of propositions, has a history in

philosophy as a candidate for the definition of knowledge. It has seemed unsatisfactory, because a person could hardly be said to know all the implications of a set of propositions. However, its position within an explicit processing theory presents quite a different situation. Here, having knowledge is distinguished from having it available for any particular use, and in a principled way that depends on the details of the processing system. This formulation in fact corresponds to the actual use of the term in artificial intelligence, where it is necessary to talk about what is available in a data structure that could be extracted by more or different processing.

7.2. Obstacles to Consideration

The basic results we have been reviewing have been with us for twenty years in one guise or another. Some attitudes about them have grown up that are obstacles to their correct interpretation. These are worth mentioning, at least briefly:

- *The Turing Tar Pit.* The phrase is Alan Perlis's.[8] The view is that all distinctions vanish when considering systems simply as universal machines (ie, as Turing machines), since all systems become equivalent. Therefore, general results about universality cannot be of interest to any real questions. On the contrary, the question of interest here is precisely what structure provides flexibility. The discovery that such flexibility requires symbols is a real one. The Turing Tar Pit only traps the unwary who already live within the world of universal symbol systems, which of course computer scientists do.

- *The computer as tool kit.* The universality of the digital computer means it can be used to simulate and build models for any system of interest, from chemical processing plants to traffic control to human cognition. Therefore, its role and significance is no different for cognitive science than for any other science or engineering. On the contrary, it is the structure of the digital computer itself (and the theoretical analysis of it) that reveals the nature of symbolic systems. When the computer, as a general purpose tool, is used to simulate models of mind, these are models of symbol systems (though of different architectures than that of the computer being used as tool).

- *The requirement for unbounded memory.* Universality implies unbounded memory. All real systems only have bounded memory. Therefore, the property of universality cannot be relevant to the understanding of intelligent mechanisms. On the contrary, as we emphasized earlier, the structural requirements for universality are not dependent on unbounded memory, only whether the absolute maximal class of input-output functions can be realized. Symbol systems are still required if universality is demanded over any sufficiently large and diverse class of functions.

- *The ignoring of processing time.* Universality requires no restraint on processing time. Indeed, simulations run indefinitely slower than what they simulate. But time and resource limits are of the essence of intelligent action. Therefore, universality results are of little interest in understanding intelligence. On the contrary, the requirement for symbol

[8] Some readers may be unacquainted with the famous Tar Pits of La Brea California, which trapped and sucked down innumerable prehistoric animals, without distinction -- large and small, fierce and meek.

systems remains with the addition of physical limits, such as real time (or reliability, sensitivity, ...). The objection confuses necessary and sufficient conditions. The real question is what is the subclass of symbol systems that also satisfy the real time constraint. This is sufficiently important to the general argument of this paper that we take it up below in more detail.

- *The requirement for experimental identification.* An experimental science of behavior can only be concerned with what it can identify by experimental operations. Universal machines (and various general representations) mimic each other and are indistinguishable experimentally. Therefore, they are not of interest to psychology. On the contrary, if humans have this chameleon-like character (which it appears they do), then it is the basic task of psychology to discover ways to discern it experimentally, however difficult. Without downplaying these difficulties, the objection overstates the lack of identifiability in the large (ie, in the face of sufficiently wide and diverse contexts and varieties of measurement).

- *The uniform nature of symbol systems.* General symbol systems imply a homogeneous set of symbols, in which everything is done by uniform mechanisms. But physiology and anatomy show clearly that the nervous system is filled with computational systems of immense specialization (and evolution confirms that this is how it would be). Therefore, humans (and other animals) cannot have symbol systems. On the contrary, this objection inducts the wrong attributes from existing computer architectures. The functional properties we have summarized are what is important. These can be realized in an immense diversity of schemes, including ones that are highly parallel and full of special mechanisms.

- *The discrete nature of symbols.* Symbol systems are ultimately just a collection of bits -- of yes's and no's. Such a discrete representation cannot possibly do justice to the nature of phenomenal experience, which is continuous and indefinitely rich. On the contrary, there is good reason not to trust the intuition about the relation of phenomenal reality and discreteness. On the side of constructed systems, speech and vision recognition systems begin to show adequate ways in which continuous environments can be dealt with. On the side of the human, the discrete cellular nature of biological systems (and below that of molecular structure) gives pause on the anatomical side; as does the sampled-data character of the visual system on the behavioral side. However, nothing to speak of is known about "continuous" symbol systems, ie, systems whose symbols have some sort of continuous topology.

- *The computer metaphor.* The computer is a metaphor for the mind. Many metaphors are always possible. In particular, new technologies always provide new ways to view man. Therefore, this metaphor too will pass, to be replaced by a metaphor from the next technology. On the contrary, though it is surely possible, and sometimes fruitful, to use the computer metaphorically to think about mind, the present development is that of a scientific theory of mind, not different in its methodological characteristics from scientific theories in other sciences. There has been an attempt in the philosophical literature to take *metaphor* as a metaphor for all theory and science (Black, 1962), a view well represented by Lakoff (1979) at this conference. Like all metaphors, it has its kernel of truth. But the sign is wrong. The more metaphorical the less scientific. Again, the more metaphors the better, but the more comprehensive the theory of a single phenomenon, the better. *Computational metaphor* does not seem a happy phrase, except as a rhetorical device to distance theoretical ideas flowing from the computer and keep them

from being taken seriously as science.

7.3. The Real-time Constraint

A brief discussion of the constraint that processing occur in real time may serve to clarify the role of symbol systems in the total endeavor to understand the phenomena of mind.

No doubt, living in real time shapes the nature of mind, and in more ways than we can imagine at present. For instance, it produces the existential dilemma that gives rise to search as a pervasive feature of all intelligent activity. Limited processing resources per unit time continually must be committed *now* without further ado, the opportunity to spend *this* now already slipping past. Imperfect present knowledge always produces imperfect commitments, which leads to (still imperfect) corrective action, which cascades to produce combinatorial search.

As noted earlier, such considerations do not remove the need for symbols. Intelligent activity in real time cannot be purchased by foregoing symbols. Rather, those symbol systems that can perform adequately in real time become the focus of interest in the search for a theory of mind. How would one seek to discover such a class? One way -- though only one -- is to work within the class of symbol systems to find architectures and algorithms that are responsive to the constraints of real time.

An example is the intensive explorations into the nature of multiprocessing systems. This is being fueled much more generally by computer science interests, driven by the advances in technology which provide increasingly less expensive processing power. The range of such explorations is extremely broad currently and much of it appears remote from the interests of cognitive science. All of it assumes the total systems will be general purpose computers (though with interesting twists of efficiency and specialization). It will add up eventually to a thorough understanding of the space, time and organization trade-offs that characterize computers that operate under severe time constraints.

Another example, somewhat closer to home, are the so called *production systems* (Waterman & Hayes-Roth, 1978), which consist of a (possibly very large) set of condition-action rules, with continuous parallel recognition of which rules are satisfied in the present environment and selection of one (or a few) of the satisfied rules for action execution. There are several reasons for being interested in such systems (Newell, 1973, Newell, 1980). However, a prime one is that they are responsive to the real-time constraint. The parallel recognition brings to bear, at least potentially, all of the knowledge in the system on the present moment when a decision must be made. Such systems are also universal symbol systems. They would have done as well as SS for illustrating the nature of symbols, save for the confusion engendered by their also exhibiting aspects responsive to other constraints.

The point of both examples (and others that could have been given) is not the particular contributions they might make individually. Rather, they illustrate the ability to explore classes of systems that are responsive to additional constraints, by developing subclasses of architectures within the class of universal symbol systems. That the space of all universal symbol systems contains vast regions of systems inappropriate to some of the other conditions of mind-like systems is irrelevant. More precisely, it is irrelevant if the larger class is a suitable base for further analysis and exploration -- which is exactly what current experience in computer science attests.

8. CONCLUSION

Let us return to our general problem of discovering the nature of mind, and the decomposition of that problem into a dozen constraints (Figure 1-1). We now have a class of systems that embodies two of the constraints: universality and symbolic behavior. Furthermore, this is a *generative* class. It is possible to construct systems which are automatically within the class. Thus this class can be used to explore systems that satisfy yet other constraints. Indeed, that is exactly the twenty-five year history of artificial intelligence -- an explosion of exploration, all operating from within the class of systems that were automatically universal and symbolic. The generative character comes through clearly in this history, as the initial versions of digital computers were shaped via the development of list processing to also bring their general symbolic character to the fore.

This class of universal-symbolic systems is now tied to a third constraint, rationality. That is what the Physical Symbol System Hypothesis says. Unfortunately, the nature of rational action is not yet well enough understood to yield general generative formulations, to permit exploring other constraints within a constructive framework that automatically satisfies the rationality constraint (as well as the universality and symbolic behavior constraints). Major attempts in artificial intelligence still start from basic symbolic capability and posit their own idiosyncratic processing organization for attaining rational behavior. However, some parts of the puzzle are already clear, such as the notion of goal and goal hierarchies, and the concept of heuristic (ie, knowledge controlled) search. Thus, we may not be too far away from the emergence of an accepted generative class of systems that are universal-symbol and also rational. The excitement that rippled through the artificial intelligence world at the beginning of the seventies when the so-called planning languages first came on the scene (Hewitt, 1971, Rulifson, Derksen & Waldinger, 1972), stemmed in large part because it seemed that this step had been taken. We didn't quite make it then, but experience keeps accumulating.

This phenomenon continues: Discovering that it is possible to shape new subclasses that satisfy additional constraints on our list. We discussed briefly the real-time constraint. We did not discuss, but could have, progress with respect to a few of the other constraints (though by no means all), eg, linguistics or vast knowledge -- not just general progress, but progress in shaping a generative class of systems that automatically by construction satisfies the constraint.

I end by emphasizing this evolution of generative classes of systems that satisfy successively more constraints in our list, because it can stand as a final bit of evidence that we are on the right track -- that symbol systems provide us with the laws of qualitative structure within which we should be working to make fundamental progress on the problem of mind. It is one more sign, coupled with the rich web of concepts illustrated in the prior pages, of the scientific fruitfulness of the notion of a physical symbol system.

Francis Crick, in his Danz lectures, *Of Molecules and Men* (1966, p7), discusses the problem of how life could have arisen:

> [This] really is the major problem in biology. How did this complexity arise?
>
> The great news is that we know the answer to this question, at least in outline. I call it news because it is regrettably possible in very many parts of the world to spend three years at a university and take a university degree and still be largely ignorant of the answer to this, our most fundamental problem. The answer was given over a hundred years ago by Charles Darwin and also by A. R. Wallace. Natural selection, Darwin argued, provides an 'automatic' mechanism by which a complex organism can survive and increase in both number and complexity.

For us in Cognitive Science, the major problem is how is it possible for mind to exist in this physical universe. The great news, I say, is that we know, at least in outline, how this might be. I call it news because, though the answer has been with us for over twenty years, it seems to me not to be widely recognized. True, the answer was discovered indirectly while developing a technological instrument; and key steps in its evolution occurred while pursuing other scientific goals. Still, there remains no reason not to face this discovery, which has happened to us collectively.

9. REFERENCES

Allport, D. A. Conscious and unconscious cognition: A computational metaphor for the mechanism of attention and integration. In Nilsson, L.G. (Ed.), *Perspectives on Memory Research,* Hillsdale, N.J.: Erlbaum, 1979.

Ashby, W. R. *Introduction to Cybernetics.* New York: Wiley 1956.

Black, M. *Metaphors and Models.* Ithaca, N.Y.: Cornell University 1962.

Brainerd, W. S. & Landwebèr, L. H. *Theory of Computation.* New York: Wiley 1974.

Church, A. An unsolvable problem of elementary number theory. *The American Journal of Mathematics*, 1936, 58, 345-363.

Clark, H. & Clark, E. *The Psychology of Language: An introduction to psycholinguistics.* New York: Harcourt Brace Jovanovich 1977.

Crick, F. *Of Molecules and Men.* Seattle, Washington: University of Washington Press 1966.

Feynman, R. P., Leighton, R. B. & Sands, M. *The Feynman Lectures in Physics.* New York: Addison Wesley 1963.

Geschwind, N. Neurological knowledge and complex behaviors. In Norman, D. A. (Ed.), *La Jolla Conference on Cognitive Science,* Program in Cognitive Science, UCSD, 1979.

Hewitt, C. *Description and Theoretical Analysis (using Schemata) of Planner: A language for proving theorems and manipulating models in a robot.* PhD thesis, MIT, January, 1971.

Hintikka, J. *The Intentions of Intentionality and other New Models for Modality.* Dordrecht, Holland: Reidel 1975.

Hopcroft, J. E. & Ullman, J. D. *Formal Languages and their Relation to Automata.* Reading, MA: Addison-Wesley 1969.

Kripke, S. Semantical analysis of modal logic II. In Addision, J. W., Henkin, L. & Tarski, A. (Ed.), *The Theory of Models,* Amsterdam: North Holland, 1972.

Lachman, R., Lachman, J. L. & Butterfield, E. C. *Cognitive Psychology and Information Processing: An introduction.* Hillsdale, N.J.: Erlbaum 1979.

Lakoff, G. Toward an experientialist philosophy: The case from literal metaphor. In Norman, D. A. (Ed.), *La Jolla Conference on Cognitive Science,* Program in Cognitive Science, UCSD, 1979.

Lindsay, P. & Norman, D. *Human Information Processing: An introduction to psychology, 2nd ed.* New York: Academic 1977.

Minsky, M. *Computation: Finite and infinite machines.* Englewood Cliffs, N.J.: Prentice-Hall 1967.

Neisser, U. *Cognition and Reality.* San Francisco: Freeman 1976.

Newell, A. & Simon, H. A. *Human Problem Solving.* Englewood Cliffs: Prentice-Hall 1972.

Newell, A. & Simon, H. A. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, 1976, 19(3), 113-126.

Newell, A. Discussion of the session on integration in information in the nervous system. In *Proceedings of the International Union of Physiological Sciences, III,* International Union of Physiological Sciences, 1962.

Newell, A. Production systems: Models of control structures. In Chase, W. C. (Ed.), *Visual Information Processing,* New York: Academic Press, 1973.

Newell, A. Harpy, production systems and human cognition. In Cole, R. (Ed.), *Perception and Production of Fluent Speech,* Hillsdale, N.J.: Erlbaum, 1980.

Nilsson, N. *Principles of Artificial Intelligence.* Palo Alto, CA: Tioga 1980.

Palmer, S. E. Fundamental aspects of cognitive representation. In Rosch, E. & Lloyd, B. B. (Ed.), *Cognition and Categorization,* Hillsdale, N.J.: Erlbaum, 1978.

Rulifson, J. F., Derksen, J. A. & Waldinger, R. J. *QA4: A Procedural Calculus for Intuitive Reasoning.* Technical Report 73, Artificial Intelligence Center, Stanford Research Institute, 1972.

Rumelhart, D. E. *Introduction to Human Information Processing.* New York: Wiley 1977.

Waterman, D. A. & Hayes-Roth, F., (Eds.). *Pattern Directed Inference Systems.* New York: Academic Press 1978.

Whitehead. *Symbolism: Its meaning and effect.* New York: McMillan 1927.

Wilson, E. O. *Sociobiology: The new synthesis.* Cambridge, Mass.: Harvard University Press 1975.

Winston, P. *Artificial Intelligence.* Reading, MA: Addison-Wesley 1977.

Yovits, M. C. & Cameron, S. (eds.). *Self Organizing Systems.* New York: Pergamon 1960.

Yovits, M. C., Jacobi, G. T. & Goldstein, G. D. (eds.). *Self Organizing Systems 1962.* Washington, D.C.: Spartan 1962.