COMPUTATIONAL EPISTEM0L06Y

Aaron Stoman

1982

Cognitive Science Research Paper

Serial No: CSRP 011

The University of Sussex
Cognitive Studies Programme
School of Social Sciences
Falmer
Brighton BN1 9QN

Aaron Sloman
Cognitive Studies Programme
School of Social Sciences
University of Sussex

COMPUTATIONAL EPISTEMOLOGY

To appear in proceedings of the Seminar on Genetic Epistemology and Cognitive Science, Fondations Archives Jean Piaget, University of Geneva, 1980.

This is an edited transcript of an unscripted lecture presented at the seminar on Genetic Epistemology and Artificial Intelligence, Geneva July 1980. I am grateful to staff at the Piaget Archive and to Judith Dennison for help with production of this version. I apologize to readers for the remnants of oral presentation. Some parts of the lecture made heavy use of overlaid transparencies. Since this was not possible in a manuscript, the discussions of learning about numbers and vision have been truncated. For further details see chapters 8 and 9 of Sloman [1978].

I believe that recent developments in Computing and Artificial Intelligence constitute the biggest breakthrough there has ever been in Psychology. This is because computing concepts and formalisms at last make it possible to formulate testable theories about internal processes which have real explanatory power. That is to say, they are not mere re-descriptions of phenomena, and they are precise, clear, and rich in generative power. These features make it much easier than ever before to expose the inadequacies of poor theories. Moreover, the attempt to make working programs do things previously done only by humans and other animals gives us a deeper insight into the nature of what has to be explained. In particular, abilities which previously seemed simple are found to be extremely complex and hard to explain - like the ability to improve with practice.

The aim of this "tutorial" lecture is to define some very general features of computation and indicate its relevance to the study of the human mind. The lecture is necessarily sketchy and superficial, given the time available. For people who are new to the field, Boden [1977] and Winston [1977]. The two books complement each other very usefully. Boden is more sophisticated philosophically. Winston gives more technical detail.

I speak primarily as a philosopher, with a long-standing interest in accounting for the relation between mind and body. Philosophical analysis and a study of work in AI have together led me to adopt the following neo-dualist slogan:

Inside every intelligent ghost there has to be a machine.

According to the old idea of dualism the human body is a machine and inside it there is some kind of spiritual extra, scornfully referred to by Gilbert Ryle as "the ghost in the machine". The new dualism turns

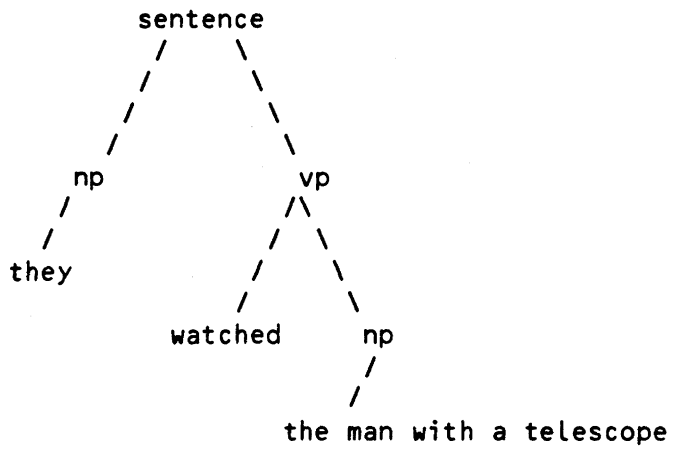
this on its head. But what sort of machine are we saying must be in every intelligent ghost? My answer is: a computational machine. Most of this talk is an attempt to explain the meaning of the claim, and to sketch some of the implications and problems arising from it.

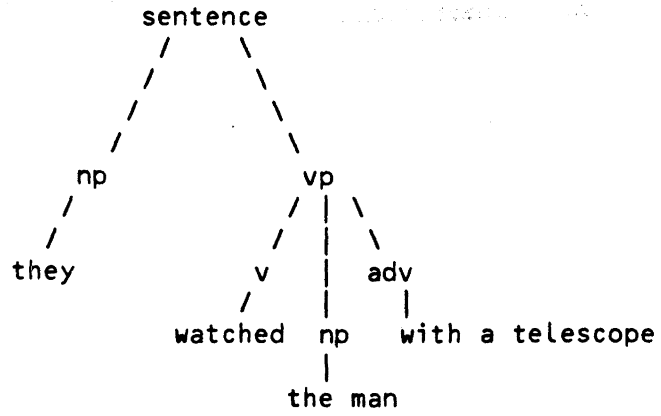
What is computation? It is rule-governed structure manipulation (sometimes referred to as symbol manipulation). That raises two questions, "what is a structure", and "what is rule-governed manipulation?" I shall try to give a sketch of an answer to each, but I think it is very important to stress that we do not know the answers. We have some examples, but I feel we are on the beginning of a long road with much to explore. There are very many sorts of structures and many sorts of manipulations of structures, but we have begun to understand only a small subset. All I can do now is gesture toward some of that sub-set. (Mathematically minded logicians and computer scientists study an even smaller subset.) Part of the aim is to reveal how inadequate is the conception of computation of most people, including many who use computers as a tool for processing experimental data.

What are structures? The most general answer is that anything is a structure which has parts with properties and relationships. To give some flesh to this idea I shall start by giving some examples of structures and I hope you will find most of them familiar. Since almost anything can be a structure, there is something very weak about saying computation is the manipulation of structures. What gives content to the claim is the detailed work on various sorts of structures, exhibiting varied forms of computation.

Examples of structures

We start with a familiar example from modern linguistics. A string of words can be interpreted as a meaningful sentence in more than one way.





The different interpretations may be represented by two tree structures. The sentence "they watched the man with a telescope" is syntactically and semantically ambiguous. There are two interpretations, in which we grasp the sentence as having two distinct structures, represented approximately by the diagrams above. On the first interpretation the man watched has the telescope, on the second the watchers.

These are fairly obvious and familiar examples of structures. There exist computer programs which can take English sentences and build structures like those above, in the computer. The computers don't necessarily produce diagrams on two-dimensional surfaces (although some do print out diagrams). Rather, they build internal symbolic structures which in some sense reflect what we see in the diagram. For example, here is a small example of an interaction with a demonstration program used for teaching students at Sussex University (on a PDP11/40 mini computer). Lines beginning with a colon are typed by the user, the rest by the computer.

```
: parse([they watched the man with the telescope]);
** [s [np [pn they]]
      [vp [vnp watched]
          [np [snp [det the] [qn [noun man]]]
              [pp [prep with]
                  [np [snp [det the] [qn [noun telescope]]]]]]]]

: parse([they took the telescope from the man]);
** [s [np [pn they]]
      [vp [vnpfromnp took]
          [ppnpfromnp [np [snp [det the] [qn [noun telescope]]]]
                      from
                      [np [snp [det the] [qn [noun man]]]]]]]
```

The two-dimensional symbols printed out on paper are here isomorphic with labelled tree diagrams. The structure inside the computer is something different again: a tree or network structure is imposed on an essential linear memory by allowing some of the contents of the memory to be symbols (usually bit-patterns) interpreted by the computer as representing other locations. Of course, there is no reason why computers have to be designed with such linear memories. In fact, the linear structure is itself an abstraction resulting from the way a network of physical connections is interpreted by the computer.

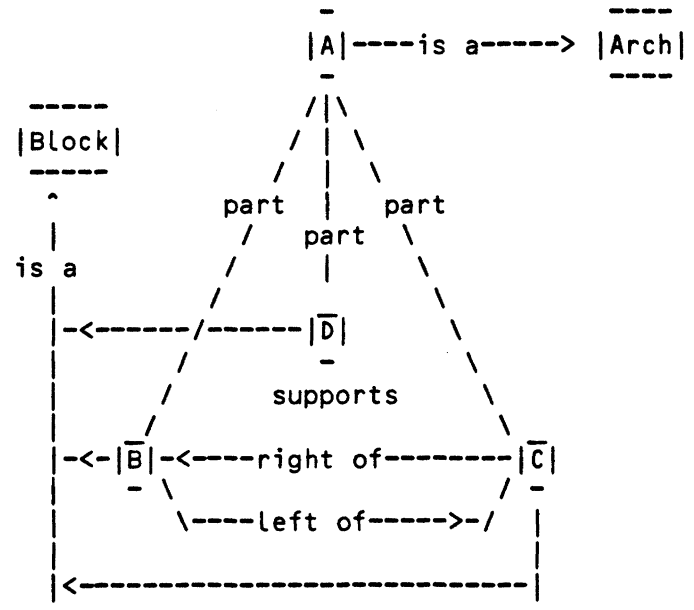
The processes of understanding a sentence are far more complex than the processes which merely build a structural description. Computer programs which are capable of making sense of a piece of coherent narrative must be able to build elaborate internal structures representing the system's cumulative grasp of the total situation described by sentences already interpreted.

Another, more familiar, example of a structure is a bit-pattern, for example:

ABABBAABBBBAAABA
1010011000011101

Each of these is a bit pattern, an ordered set of "binary" symbols. Each element of the sequence is a member of a set of two possible alternatives, often 1 or 0. It could just as well be a square and a circle. In computers such bit patterns are used, for example, to represent numbers, to represent instructions and to represent locations in the memory of the computer. As we shall see, the same patterns can be given different interpretations, depending on the operations performed on them.

Here is rather more complicated example which you will also find in Winston's and Boden's books. It is based on Winston's Ph.D thesis, reprinted in his collection The Psychology of Computer Vision. Consider an arch with two supports and a pedestal across the top. Winston suggests that if a program or person is to be able to develop new concepts from being presented with examples of instances and non-instances, it will be necessary for the program to build internal



structures something like this: This is meant to be a network which is made of nodes linked to other nodes by arcs, and the arcs have "labels" which indicate what sort of relationship is intended. Each node represents some kind of entity, abstract or concrete, and the arcs carry information about their properties and relations. The node A - representing the arch - is linked to three nodes, B, C and D by arcs which are labelled "has part", so the thing A is represented as having three parts. This captures some of the gross structure of the arch. The parts themselves are linked to the concept of a block, a generic block,

by nodes labelled "is a", so B is a block, C is a block, and so on. They are linked to each other by arcs which may have labels like "supports" so B supports D and C also supports D. Clearly the same method of representation in a network can be used to express many other properties and relationships, provided that we already have symbolic names for them.

Winston talks about building very rich networks of that kind and shows, at least in outline, how to make a computer do it, when presented with a certain class of line-drawings. The network would be a structure, and construction and use of the network in recognising objects would be examples of manipulation of structures, i.e. computation. It is perhaps worth noting that in order to do this the program has to treat the input image, i.e. the picture of an arch, as a structure. Analysis of the image to find substructures such as lines, junctions, regions, etc. is also computation. The original object in the world (e.g. the arch made of bricks) is also a structure. Manipulation of such 'external' objects can also be regarded as computation. For instance, moving bricks around on a table might be part of a computation deisgned to find the best arrangement of the furniture in the room: the bricks are taken to represent larger objects.

Notice that although writing the programs is a non-trivial exercise, there is nothing essentially mysterious about the process of constructing such abstract network representations within a mind or computer. There exist programs already which, when presented with pictures of three dimensional objects will do an analysis, that is will decompose the image into parts, will categorize the parts, test for the relationships between the parts and build inside the computer networks like that above and then store them in some larger network. The program may even interpret the 2-D image in terms of a quite different set of 3-D structures. That is, given a 2-D configuration it creates within itself a structure representing a 3-D object, a cube, for example. Such a system will not just find the name 'cube', but might, for example, build a network representing each of the visible faces, edges and corners, and their relationships. It might even represent some of the 'inferred' invisible parts.

There are many unsolved problems about how such stored structures will later be retrieved, what kinds of indexing mechanisms are needed, problems about how to match a stored network against the network constructed from a new image, and many problems about how exactly new experiences should cause modifications to stored representations. Winston talks about a matcher which, the next time it sees something a bit like the original picture but slightly different, say an arch with a triangular block on the top, builds a network, tries to match the two networks and finds that they are partly the same and partly different. So the program has to build a third network representing the similarities and the differences between the first two. So structure-manipulation includes building structures which represent properties and relations of other structures.

Here is another little example which is meant to be not a structure that represents another static structure like a sentence or an arch but a structure that represents a type of action.

```
PROCEDURE([GRASP ?X])
   PRECONDITION: CLEARTOP X
   PRECONDITION: EMPTY HAND
   MOVETO X
   GRASP
END
```

This structure is a piece of program that might be part of a goal-directed system using a suitable programming language. It defines a strategy for grasping some unspecified object, represented by the variable X. There are two pre-conditions, defining sub-goals which may have to be achieved: (a) clear the top of X to make sure that your hand can come down from above in order to grasp it and (b) make sure that the hand is empty. When those pre-condition are satisfied, perform the action of moving the hand to X and then grasp.

This is not meant to be taken seriously as a theory about what goes on in a child (or robot) able to grasp bricks of many shapes from different angles and so on, but just to illustrate a structure, a symbolic structure which may play a role in the computation involved in forming a plan or performing an action. The same structure may have two very different roles: in one role it functions as a program. This means some other program or machine (usually called an interpreter) examines portions of it and performs actions as a result. In another role, instead of being a program, the set of symbols can function as a structure created by or modified by other programs, like Winston's nets. For instance, G.J. Sussman A Computational Model of Skill Acquisition describes how such programs could be synthesized and later executed by a learning and planning program which initially has only a simpler set of action plans. While the program is being synthesised, or when it is being modified because something has gone wrong, it is treated as a structure manipulated by other programs.

This duality of program and structure is quite common in computing. A structure which is built up and compared with other things and looked at as an object with parts and relationships may itself have another role where it generates behaviour because it is treated as a program. For example a compiler is usually a program which takes a program written in some language convenient for people to use, analyses it, and builds a machine-language program executable by the computer. The 'compiled' program thus starts life as a manipulated structure. Later, when run, it generates behaviour. I will return to this structure/procedure duality later. It is one of several examples of how computational ideas upset our ordinary ways of categorising things. I have tried elsewhere to show how they also upset traditional ways of thinking about mind and body, about what a machine is, and the traditional distinction between a free agent and a deterministic machine.

The idea of manipulation of symbolic structures is very old. It is explicit in the aristotelian conception of logical inference as making use of valid schemas, such as:

```
ALL A ARE B
ALL B ARE C
THEREFORE: ALL A ARE C
```

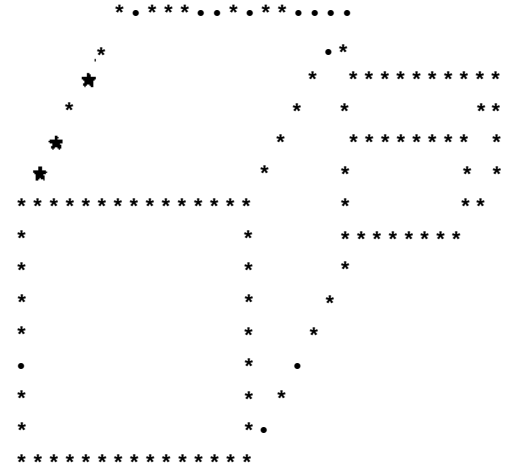
More modern logic replaces such symbols with new notation involving quantifiers and functions applied to arguments (following Frege), but

these are no less manipulable structures. In fact some have claimed that there is nothing more to computation than deduction, though it is clear from our present standpoint that deduction, that is logical manipulation of logical symbols, constitutes a special case of computation.

Chemistry also provides old and familiar examples of symbolic structures which may be manipulated in reasoning about events in the world. For instance symbols representing the structures of molecules are often used to help us understand chemical transformations, e.g.

<H-H> + (H-H) + (0=0) –> (H-O-H) + (H-O-H)

Visual perception starts from one or two 2-D array-like structures, namely retinal images consisting of a manifold of measures of features like intensity, colour and perhaps optical flow. Much simpler array-like structures can also be treated as if they were visual images. Here is a very simple example of a two dimensional array type of structure with cells which may have one of two properties: occupied or empty.



Work has also been done on much richer visual arrays. The information may be quite complicated, including colour, intensity, intensity gradient, velocity flow measures, or the stereo discrepancy between left and right images. So there can be quite a lot of information at each location and it is part of the task of a visual system to build up some kind of meaningful interpretation of that information in terms of an external world containing objects and events, with properties and relationships. Philosophical sceptics have argued that there is no rational way of doing this. The design of intelligent robots will prove them wrong – or at least reveal a conception of rationality which they had not grasped.

The information produced by a visual system needs to be represented in a structure which can be used in guiding actions, planning, and various kinds of learning. Perhaps networks of the type described by Winston would be useful. But many alternative structures need to be explored. My own work on image interpretation (reported briefly in chapter 9 of my book) indicates that in a visual mechanism able to perform the tasks performed by human and many animal visual systems, it is necessary to construct many different sorts of structures in parallel, concerned with different aspects of images and scenes. That is visual perception involves simultaneous but co-operative computations concerned with different domains of structures. The co-operation is

achieved through 'message-transmission[1]. That is structures representing partial results of analysis, or queries, are created by one sub-process and made accessible to another, possibly by interrupting it, possibly just leaving it to examine the new structure when it reaches appropriate instructions in its own program.

So far I have been presenting a more or less arbitrarily selected subset of examples of types of structures which may be involved in computation of one sort or another. I have been mainly concerned to demonstrate that there is a *\/ery* rich variety of types of computations. What is essential to all the structures we have discussed is that they are composed of parts, with properties and relationships. Computation involves addressing those parts, and checking or changing the properties and relationships, possibly by deleting or adding parts. In some cases these parts properties and relationships *are* taken to have a meaning, i.e. they are interpreted as representing parts, properties and relationships of something else. But fundamental to their role in computation is the fact that independently of any such interpretation the structures themselves can be manipulated. In the course of giving examples, I have indicated briefly some of the kinds of things which may be done with structures, illustrating what I mean by 'manipulation[1]. Here is a reminder of some types of manipulation of structures included under the concept of 'computation[1] (some elements of the list overlap with others):

MATCHING (testing similarity and/or building a representation of similarities and differences)
CONSTRUCTION or EXTENSION
COPYING
MODIFICATION (replacing a part, or changing a relationship)
STORING (inserting one structure in a larger one)
SEARCHING (examining a large structure to find a part)
SORTING (reordering components according to some criterion)
SUBSTITUTION (systematically replace all parts of a certain type with some other structure - e.g. binding variables)
INDEXING (building a new structure to facilitate searches for old ones)
OBEYING (e.g. treating a structure as an instruction or program)
PARSING (distinguishing parts and building a new structure to represent their properties and relationships)
INTERPRETING (i.e. using analysis of one structure to build a representation of something else)
TRANSLATING (building a new structure with the same interpretation)
MONITORING (e.g. waiting for a change to occur, then taking some action)
GEOMETRICAL TRANSFORMATIONS (e.g. rotation, permutation, stretching)
SENDING MESSAGES BETWEEN SUB-PROCESSES

It has been conjectured that all forms of symbol manipulation can be mapped into a relatively small sub-set, of the kind studied in recursive function theory or mathematical logic. I am not sure how important that conjecture is. The differences between different sorts of computation may be more important than their common underlying ability to be represented in a certain way. If the conjecture is true, then any kind of computation that can be done at all can be performed on a modern digital computer, provided it is made big enough and fast enough. But even if the conjecture is false, e.g. because some kinds of structures and their manipulation (e.g. continuous deformations of geometrical shapes) cannot be so represented, this may not undermine the

universality of the general notion of structure manipulation as a basis for understanding mental processes. We need to retain an open mind on these issues.

"Manipulation" may not be the best word for what I have been discussing. We think of manipulation as changing a structure but I have been using the word as a blanket term to cover many more processes including searching matching and describing. It might be better to say manipulation is a sub-species of something more general but I do not know any suitably general term except for •computation[1] itself! Suggestions would be welcome.

Meaning

I have already alluded to the difference between purely •syntactic[1] operations like comparing and describing structures on the one hand and the 'semantic[1] notion of interpreting a structure, that is treating it as representing something else. I now want to say a little about the pre-conditions for treating structures from a certain class as representing something, partly because I feel that some of the discussions, at this conference, of whether a machine could use symbols with a meaning have been unsatisfactory. What I have to say is also unsatisfactory: I feel it is just a beginning of a more adequate analysis, and I would be grateful for help.

The main idea is that in order to treat something as a certain sort of representation you have to have a class of operations which you can perform on it, and what sort of interpretation you are giving it will be determined by which sorts of operations you can perform on it. (You need not actually perform them.) The same goes for a machine. I will now begin to illustrate the way in which a class of operations performable on a set of structures is relevant to the way the structures can be interpreted. In this sense semantic processes rest on syntactic ones.

First a very simple example: I mentioned bit patterns earlier. They are structures which can be manipulated in different ways, giving them different interpretations. A very common class of operations on bit patterns is the set of boolean transformations, often used in computers. For instance, the two patterns 11001 and 10011 can be 'ANDed' together, yielding the pattern '10001', or they can be 'ORed' together, yielding '11011', or they can each be 'negated', yielding '00110' and '01100', respectively. These operations may be used as a basis for interpreting a bit pattern as a representation of a set of things. For instance, a string of two hundred bits can represent a set of integers in the range 1 up to 200, the N'th bit being 1 if the integer N is in the set, otherwise 0. Thus, if the string begins '1101001', then the integers 1,2,4 and 6 will be members of the set. The operation of 'ANDing' then represents the formation of the intersection of two sets, while 'ORing' represents the union and 'Negating' a bit pattern represents finding the complement of the set.

Notice that it is not essential that the symbols 1 and 0 be used with the roles illustrated. Their roles could be reversed, so that, for example, the AND of '11001' and '10011' would then be '11011'. In this case 0 would represent membership of the set, and 1 not. The intrinsic character of the symbols used is unimportant: it is the role they play

in the various operations that matters.

There are many other operations on bit-patterns used in computers, including shifting a certain number of steps to the right or left, removing a specified subset of bits, and binary arithmetic operations, which play a particularly important role in the use of computers for doing calculations with integers. For example the 'successor' operation starts with a bit pattern, and if the right-hand element is a 0 replaces it with a 1, otherwise it replaces the 1 with a 0 and starts again on the remaining pattern to the left. Similarly addition and subtraction of two numbers can be represented simply by 'syntactic' operations on these structures. It is because the computer has such operations programmed in to it that we can say that it interprets bit patterns as numbers. But we have seen that other sorts of operations on bit patterns enable computers to give them different interpretations.

There are still more operations which digital computers perform on bit patterns some of which involve using a bit pattern as a representation of on the one hand a machine instruction and on the other hand a location in the computer.

This illustrates my general point that how something is interpreted by X, what meaning is assigned to it by X, depends on the class of operations (manipulations) that X can perform on it. It does not matter whether X is a person or a machine: the principle is the same. And in general, for any sort of structure there will be infinitely many different sorts of interpretations corresponding to different classes of manipulations. This is as true of pictures as of sentences or mathematical formulae.

The interpretation of symbols found in logic text books provides another example. If a system which is given the formula "P -> Q", and later the formula "P", then constructs and stores the formula "Q" then we have reason to believe it is interpreting the first formula, roughly, as "If P then Q". Similar remarks can be made about the following inference rule:

P -> Q; not-Q; Therefore not-P.

Or, in English,

If P then Q. Not Q. Therefore not P.

In fact, somewhat more than this is required if we are to say that the machine gives the formulas their ordinary logical interpretation: the system must make use of formulas in controlling its actions, making choices, interpreting sensory stimulation, and so on. Otherwise there is no basis for saying that it accepts the conclusions as <u>true</u>, and therefore no reason to say that it is making logical inferences.

Once you get into formulas which express generality you are in a whole new world of power and glory and headaches. Here is a formula using the universal quantifier:

Ax (Px & Qx) -> Rx

Or, in English:

For Any x, if x is P and x is Q then x is R

A possible example:

If x is in Geneva and x studies Psychology then x admires Piaget

From the above formula, using the rule of 'universal instantiation', the

following can be inferred, where a is any particular individual:

If a is P and a is Q then a is R

Predicate logic provides a rather more complex set of operations on symbolic structures than some of the other operations, such as the manipulation of bit patterns. But the more complex operations can be just as precisely defined. You can check given two formulas whether one is actually a substitution instance of the other according to the rules, though this is not as easy as you might think. Anyway, here is a somewhat more complex example of the syntactic operations involved in logical inference. The third formula is obtained from the first two by 'resolving' them:

AxAy(Px & Qy) -> Rxy

Pa v Sa

Ay (Qy -> Ray) v Sa

In English:

> For all x and for all y, if Px and Qy then the relation R holds between x and y.
> Either a is P or a is S.
> Then, either for all y if y is Q then the relation R holds between a and y, or a is S.

(Once you are familiar with this sort of formalism, English seems to be much less clear, unambiguous and elegant.) Here we have a quite complex and abstract operation which involves doing a sort of match between two or more symbols, finding one is a part of another and then generating some new symbol using a precisely defined algorithm for manipulating symbols without any regard for their meaning. One can treat this logical matching operation as just a game with symbols, but because it has some very important properties (which I admit I do not fully understand and will not try to summarize) it is possible to say that if you do that kind of operation on these symbols then you are treating these symbols as belonging to a language of predicates and relations in which you can assert things that may be true or false. Of course, as natural languages show, using just this set of operations is not a necessary condition for treating symbols as expressing true or false propositions.

Logic provides another example of the point that manipulation of one class of structure may involve treating it as representing something else, as having a meaning, a significance, if that manipulation involves a suitable class of operations. The same applies to the use of so-called 'figurative', or 'analogical', or 'iconic' representations, which are often treated as if they are a very special, essentially human phenonmenon, quite distinct from computations.

Figurative representations

During the conference, there was considerable discussion of 'figurative' representations and how they compare with propositional or as some people say 'symbolic' representations. It is easy to get confused about this, for example thinking that 'figurative' symbols are essentially continuous while the others are discrete, or that digital computers cannot handle the former.

In chapter 7 of my book I discuss this matter at great length, in particular distinguishing Fregean representations and analogical representations. As far as I know, Gottlob Frege was the first logician to offer the analysis of propositional representations in terms of the application of functions to arguments, which pervades much of logic and linguistics nowadays, as well as computing. On this view, a sentence like

'The left hand of Fred is open'

would have a form something like:

Open(left_hand(Fred)).

This sort of representation has a structure which has nothing to do with the structure of the relevant bit of the world. Rather, its structure represents something more like the structure of the process of arriving at a decision whether the statement is true: identify the individual called Fred, and from there use the procedure for finding a left hand, and then use the procedure for testing whether something is open.

We can contrast Fregean operations on symbols with some of the manipulations that can be done with geometrical structures. By exploring such manipulations we can get a much clearer idea of what is involved in interpreting something as a figurative representation. For example, we can think about geometrical structures just on a line - i.e. one dimensional configurations. It is a useful exercise to try classifying the things you can do with a line or structure. You can compare two points to find out if they are the same point, or you can ask what is the distance between them, or if one is to the right of the other. If there are three points A B and C, you can check whether A is between the other two or not. You can talk about line segments on the line, and the relationships between them. You can ask: Do they overlap? Is one completely contained in another? Does one of them form exactly half of another? and so on. You can think of processes of change within a line in which one segment slides along and the configuration changes.

If you go into two dimensions it gets much more complicated. Everything you find in one dimension can of course occur within two dimensions because you can have a one dimensional line in a two dimensional space. But there is much more. In two dimensions, you can have rotations, you can have all sorts of wiggly complex shapes and many operations involving comparison of shapes, fitting shapes into other shapes, rotations, translations, stretching and so on. When you learn to treat symbols on paper as symbols of predicate calculus, for instance, you are actually using a sub-set of the kinds of shape comparisons and manipulations that one can do in two dimensions. When a thing curved this way "(" is seen as significantly different from one curved this way ")", they are treated as a pair of brackets. You have to be able to perceive the geometrical relationship between them in order to understand the bracketing conventions or to tell the difference between a "p" and a "q". You have got to be able to decompose that geometrical structure. So we use geometry in our logical formalisms.

But there is a much larger class of properties and transformations of spatial structures, and this gives such structures enormous potential for use in representing other things, not necessarily with exactly the same structure. For instance visual perception and picture interpretation involve treating a two-dimensional structure as a representation of a three-dimensional one. This power of spatial structures can be exploited no less by a computer than by a person. There is no evidence that such analogical, even intuitive, thinking is

anything but a species of the general concept of computation I have been illustrating.

As an exercise attempt to make lists of kinds of comparisons and operations you can perform on two dimensional structures. Then ask yourself, say, which sub-set of the operations is relevant to treating a two dimensional structure as a map, say of Switzerland. There are some special classses of maps for instance, maps of transport systems, which very often do not accurately represent distances and directions, but rather represent connectivity and order along the lines. Distances and directions are represented only very approximately. Ask yourself what kinds of operations you have to be able to do on that sort of structure to be able to use it as that kind of map?

One of the questions about 'figurative representations' addressed by artificial intelligence work on vision is: how can you treat a two-dimensional structure as a representation of a three-dimensional structure? Right now only a tiny sub-set of the problem has been explored, mostly to do with objects bounded by straight lines and plain surfaces -- the so-called "block's world". Work on curved surfaces is beginning to get off the ground, though there are many unsolved problems about how curved lines and surfaces should be represented in a mind or computer. There is considerable progress on the task of interpreting straight line drawings as representing three-dimensional configurations. It turns out for instance to be important to be able to distinguish classes of junctions, for instance, L junctions, T junctions, cross junctions, arrow junctions and others. The picture above representing two blocks is an example.

The class of two-dimensional structures and the class of operations on them are huge. There is an enormous space of possible structures and it is a very powerful generator of special cases with special uses. For instance, we use maps, flow-charts, time-tables, graphs, family trees, histograms, pie-charts, photographs, line-drawings, sketches and many more types of two-dimensional representations to help us store or communicate information, solve problems, explore possibilities, etc. The operations we perform on these structures, whether on paper or in our minds are all examples of the general concept of computation, defined above. What makes them seem to be different may in part be the fact that our visual systems have evolved very powerful procedures for manipulating spatial structures, so we feel we have some effortless way of doing this, as compared with, say, logical thinking. But for an expert logician, who has had a chance to build up suitably powerful logical procedures, logical inferences can feel just as natural and intuitive. In both cases as the richness of the complexity of the examples increases we can eventually reach computational limits.

For example, family trees are an often discussed example of a 'figurative' representation. More generally relationships form networks such as I presented earlier. Often two dimensions are not enough if we want the links in the net to cross over one another, representing a complex set of relationships. As a network becomes increasingly complex and tangled, it becomes less intuitively obvious how things are related. So the fact that a representation is 'figurative', or 'analogical' does not in itself guarantee that it will be easier for people to use than some other, more abstract translation.

We can illustrate the use of spatial representations in connection with the puzzles described by some of the research students at the conference. We were told about experiments with children trying to solve various puzzles involving trains, the missionaries and cannibals puzzle, the boatman puzzle. It is possible to think of the child as exploring a "space" of possible moves. Each action is movement in an abstract space, to a new location. There are different states you can move to from the initial state and from each of those states there are still more states and so on.

It would be very interesting to ask whether a child can be explicitly taught to make use of that kind of representation, and whether it would help. I tried in an informal way with one of mine at the age of about six, to help him think about the problem of choosing a move in a game like draughts or chess. It was a long process. I think he got somewhere and it was a useful tool when he played subsequent games. But it was like learning to play the piano: you cannot teach it in a week or half-an-hour, so the time-scale for the typical psychological experiment is too short. (My own 'experiment' was too informal to be useful as a source of data.)

The class of two-dimensional spatial structures, and more generally N-dimensional spatial structures, and operations thereon, is very large and there are many sub-sets that people have begun to explore, but I expect there are still more sub-sets that we do not know about. Detailed formal exploration can take the form of designing computer programs which analyse or interpret these structures and use them to solve problems. I believe this is the best way to increase our understanding of so-called figurative representations -- it is more likely to yield new insights than either laboratory experiments or introspection.

My next example is a program developed at Sussex which interprets pictures. The attached figure gives examples of the images it works on: two-dimensional configurations of dots. Sometimes without prompting, sometimes with prompting, people interpret those images as representing letters forming a word. In fact, most people seem to 'see' several different sorts of structures, not just letters and words, but also lines, junctions between lines, pairs of parallel Blines, and flat over-lapping plates. It seems that here, as in much other visual perception, there are many different classes of structures all being manipulated in parallel.

Our program, which we call Popeye (since it was written using the language POP2), is an attempt to explore the design of mechanisms capable of performing such computations. For instance, in the program there is 'knowledge' of a class of two-dimensional arrays, a class of possible configurations of lines (which may be parallel, of the same length, of different lengths, have gaps in them, etc.), a class of two-and-a-half dimensional configurations of overlapping plates (half because there is no real depth, just a 'behind' relation), a rather abstract class of structures made of strokes which may have simple relationships like meeting in the middle or at the end, or the end of one meeting the middle of another, and finally a class of structures at the "top level" which are the sequences of letters recognized in the image. A sub-set of the class of letter sequences forms the set of known words. The Popeye program uses its analysis of all these different structures in order to 'see' the word.

I think that when people are presented with these pictures, there are quite complex things going on simutaneously in all those different classes of structures- This parallel processing of different types of structures enables partial information found in connection with one structure to help resolve ambiguities in others. This illustrates that computation as understood here need not be a simple serial process, but may include a collection of concurrent processes. In fact, the number of different sub-computations going on in parallel need not be fixed: it can change according to the demands of the problem.

It can be argued that this computational architecture permits a perceptual system to cope with ambiguities, noise, messy and complex configurations, and allows the recognition of large-scale structures to be completed before all the details have been processed. This seems to be a characteristic feature of much human visíon.

In real visual perception, the situation is much more complex. We are presented not with a static black and white configuration, but with colours, continuously varying íntensities, constantly changing retinal configurations. There are not just straight lines but many curved lines and curved surfaces and explaining how it all works is very difficult. There is an enormous amount of work to be done describing what might be going on in relation to each kind of visual structure. We are nowhere near saying "we think it might be this or that, we must do an experiment to choose." I do not believe people actually know yet what kinds of questions to ask to set up experiments in this sort of domain. I think you still have to function as a philosopher and engineer for a little longer, trying out possibilities.

Vision in new-born animals

We tend to think of new born animals, human and non-human, as very limited in their cognitive abilities. However, I am amazed when I see what happens with a new-born calf or lamb or foal: within a few hours of being born they can stagger to their feet and look around. They can see the mother and then they head for the her without falling over, mostly. They find their way to the nipple and start sucking and it seems that in order to be able to do this from a wide variety of different angles they must have enormously powerful computational systems, which can decompose a three-dimensional world into structures relevant to achieving the goal of getting milk.

At first I thought that maybe they were just guided by the smell or something like the density of milk molecules in the air, but I have seen them go the wrong end of the mother, which suggests that they are going for a kind of three-dimensional structure where the two legs meet the trunk, rather than doing something much simpler and more mechanical like water flowing down a hill to get the lowest part, i.e. following chemical gradients. If that kind of power is there in a new-born lamb, what are we to think might be going on in a newborn human being which has a longer gestation period? Maybe infants are much cleverer than we think but they have not yet learnt how to drive the mechanisms to tell us what is going on?

Learning about counting

Next comes a little example which is taken from chapter 8 of my book, the chapter on numbers. I thought I would bring it in because of the remarks made by Leo Apostel about work in artificial intelligence being too static. I am not going to tell you about a learning program that actually exists because I only half understand what I am talking about and I certainly do not understand it well enough to design detailed computational programs.

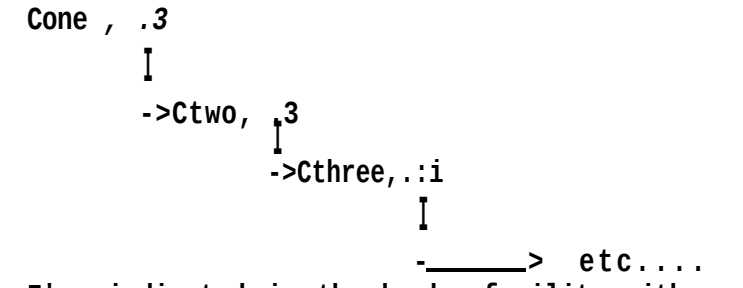
As a result of watching children when they were learning to count, between the ages of three and six, and seeing some of the things they could and could not do, I started wondering what it would be like to explain what was going on in their heads. I found when I talked to my psychologist friends who did not come from Geneva that they could not give me explanations for the phenonmena.

For instance, the kind of thing I observed would be that a child could at a certain stage count quite fluently, up to ten or eleven or more. But if I said "what comes after five?" he could not answer. He could only start at the beginning and carry on. If some of you have learnt to play the piano or musical instrument you may have that experience with a piece of music - you can start at the beginning but you cannot start at the middle. The same can apply to the reciting a poem.

Then I found that a bit later the child could answer "what comes after five?" "what comes after six?" but if I said "what comes before five?" or "Does eight come after three or before?" the answers would be random or "I do not know". However, I also found at the same stage that if he looked at the kitchen clock he could tell me the answer, using the numbers visible on its face.

So he understood the problem, he could work with a representation outside his head and perform some operations on it using his eyes and following the sequence to get the answer. But he could not do the same thing in his head. That suggested that he did not have anything like a piece of paper or screen in his head and an internal eye scanning it - it must be a different sort of representation.

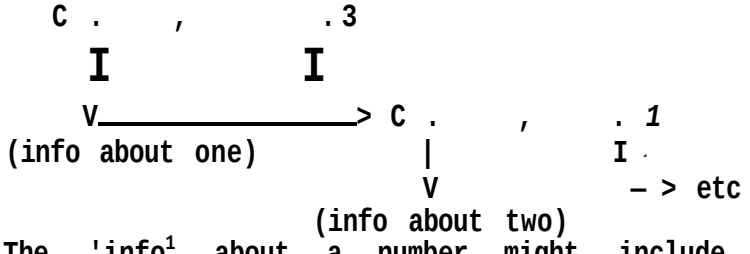
In my book I suggested that these and other phenomena might be explained by the assumption that the child builds 'list-structures' in his mind. List structures are symbolic structions composed of associative links between two pieces of information. They are commonplace in AI programs, especially those written in Lisp. A series of items of information can be built into a chain of links by making each link associate an item of the series with the next link. From each link one can then get to the next link and the item of information stored there, but one cannot find the preceding link. The use of such associative pairs to construct remembered sequences would explain the difficulty of going backwards, or answering questions of the form 'what comes before X?'



As I've indicated in the book, facility with such tasks might be acquired either as a result of building new procedures or as a result of enlarging the network of linked pairs to contain more information in the form of cross-references and backward references. For example, here is a possible procedure for answering the question "What comes before five". Look at the first of the links, store its contents and go to the next one and say "is that five?" If not, store that and go to the next link, then start again. If the next link doesn't contain five, repeat the proces. When you get to five the answer is the last number stored.

Now, it turned out that the children, and I am sure that mine are not unique, seemed to be able to work out that procedure for themselves eventually. I could not tell them how to do it, I could not feed something in through their ears to program them. They were able to work that out, making use of some general and primitive ability which seems to be there, namely to remember the last thing that said or done. I almost got the impression on one occasion that the child discovered almost by accident (though I had carefully set up the situation) that he could remember the last thing he had said when going through the number Series, and having made that discovery consciously used it as a strategy for answering 'what comes before[1]. However, I am sure that most self-re-programming is not done consciously.

Later I noticed another oddity. After a while the child did not have to think what came before five. He did not have to say "one, two, three, four, five, oh! it is four." He seemed to be able to go directly into five and work out the predecessor. So it looks as if he is able to start building a more elaborate structure that preserves information about individual numbers at each link in the chain, instead of only information about where to find the next link.



The 'info[1] about a number might include a pointer to its predecessor, for example. On this model, there is a little database of information associated with each number. I am now not going into any of the details of how that would be represented in a computer. One could do it in a language like LISP or P0P2 very easily.

Perhaps one kind of learning that goes on is that the child in order to avoid having to do elaborate computations over and over, can store results for future use by means of additional associative links in the network. That would be useful provided that another condition is satisfied, namely that the indexing mechanism associated with this network enables one, when presented with the name of the number, like

"three" to go directly into the network at the location concerned in order to find a "predecessor[11] link and say what comes before it is "two", or the "succesor" link and say what comes after it is "four".

To explain all this we'd,have to start talking about how indexing works and how you can sometimes get right into the middle of a network instead of having to chain through a long list. Clearly people do learn to do this for numbers. Probably you can all answer very rapidly with numbers up to twenty "what comes after"? or "what comes before?" You might not be able to do it so easily with the alphabet and even if you can do it with the alphabet you probably cannot do it with some poem you have learnt. With some learnt sequences you might be able to go backwards immediately on request.

There might be additional pointers into the data bases, from external networks or chains as in rapidly reciting the sequence "two four six eight..." Building a new chain of links, pointing into alternate links in the old chain might be more efficient than recomputing the series every time.

I also felt that the speed with which some people could count forwards or backwards made it look as if they could not be doing an operation which said "go to the first thing, find amongst the data base of information where the successor is, go to it then say two, then search in that base of information to find where the successor is, and then say three" and so on. It felt as if that was not the kind of thing that was happeninga when you quickly say "one, two, three, four, five, six, seven", or "a,b,c,d,e,f,g". Contrast what happens when you make use of some knowledge of a structure like a row of houses or a series of rooms in your house, and you can go from each one to the next thing but it is a much more slow and painful process where you go through some elaborate procedure for finding the next thing. The speed of expert reciters suggested that maybe they use additional chains of links pointing into the old structure.

However, these new chains are not enough to explain all the phenomena including the fact that an older child could start anywhere and apparently count on forwards. I could say "count on from five", and he would rapidly go "five, six, seven, eight". "Count backwards from eight", produces rapidly "eight, seven, six, five, four, three" and so on. You can probably do it rapidly too. The problem is that at this moment we do not have a link from three into the fast forward chain to enable you to count fowards onward from three. You could start at the beginning and keep going until you come to three and then carry on but if you need to be able to start at any point and go at speed then it would be useful to have associated with the representation of the number itself a pointer into the appropriate link into that chain. The idea is that in the database of information about the number three for example there might be information about about how to get into the chain of links for counting forward, and once you have go there you say "three" and carry on at high speed. The same applies to the fast backwards chain. Thus adding external chains can both aid speed of recitation or searching, and provide a way of recording subsets, as with the chains for "odd" and "even".

Now, what all that was meant to illustrate is that there appear to be at least three important kinds of learning which involve the further development of a structure that you have already got. One is new links

within the structure, new internal structures, illustrated by adding extra pointers, say backwards from a number to its predecessor in the original chain of number names. Another kind of learning might involve adding pointers out of the structure, for instance pointers from elements of the sequence into a concept like odd or even. You can imagine a lot of other external pointers, if you ask yourself all the things you know about the number six, for example. Thirdly, new pointers may grow into the network from other networks like the fast forward network or fast back network. The distinction between these three kinds of development is very imprecise, since it is not clear how to decide between a new cross-link in the original network and a pointer to another external network.

No doubt there are many more things to be said about types of development of an information network. One of the points I am trying to make is that there is a trade-off between extending a structure such as a network, and building new procedures for operating on that network. How one can describe a system which could make choices about when to build a new structure, when to build a new program, I do not know but it seems to me there is a fruitful field for research, here.

I do not know what kind of system could produce the succession of changes that I have been talking about. I can imagine writing a program that can do it in special cases. I suspect that in the case of a child there is something more general which has to do with learning of all sorts of things and which can be applied to this specific case of learning sequences of various sorts. This general learning mechanism can produce the kinds of results that I am talking about with numbers and with letters and many other sequences of things that we learn.

So there is lots of research to be done on this area. It could also relate to issues in philosophy which I will not elaborate on in detail, for instance questions that philosophers ask about the nature of number, the nature of mathematical discovery. I think these questions take on a new perspective if you start thinking about number concepts in the way I've suggested.

This counting example illustrates one of the points I made much earlier about the duality of programs and data. You can think of building the networks described previously as not so much the creation of some static structure, but the creation of a powerful kind of program with many options built into it. In other words what I am saying is that the distinction between program and data (or procedure and structure) can be made to evaporate if you look at it in the right way. Maybe it is better to say that the distinction is important but is not an inherent one. It depends on how the structure is used.

I want to make a last remark about that. When this sort of chain is used as a program, if you look at how we actually use it in conjunction with other programs for doing things in sequence, we notice some new computational mechanisms. For instance, in order to answer the question "how many people are sitting at this table" you have to be able to generate a sequence of pointing operations and go through that 'program' at the same time as you say, "one, two, three...." etnc. You might think that is done by following a new special program which says "say the number, point to an object, say the next number, point to the object" etc. but if you look at the way a child or even you and I can make mistakes and get out of phase and then realise you have to start again,

you see it is not a new program with first one step and then another but two programs running in parallel with some kind of monitor checking that one of them is not going too fast or too slow, and keeping them going.

I think that illustrates an important point that people in artificial intelligence are only just beginning, to address, namely that we must not think just about a single processor, a single machine running a program. We may well have to think of intelligent systems as having many machines working in parallel performing co-operative tasks and I mean really in parallel, not just a sort of simulated parallel by having first one thing take a turn and then another takes its turn. You can always get the effect by simulation provided those conditions are satisfied. But conceptually they are really parallel programs.

Final remarks

1. The need for a computational epistemology

We have begun to illustrate the multifarious ways in which structure can encode usable information, i.e. knowledge or know-how. Behaviour is applied knowledge. This is one way of looking at the difference between animals, which apparently make use of some kind of representation of the environment and their goals, and plants and purely physical systems, such as water running down a hill or a thermostat. The water has no internal representation of its being at the bottom of the hill to drive the process by which it goes round obstacles. But there isn't a simple sharp distinction between intelligent and purely mechanical processes, rather a whole range of cases.

Typically, in an intelligent system, knowledge is stored, accessed when relevant, modified when incorrect or incomplete, as well as being applied, in behaviour. These all involve the construction and manipulation of symbolic structures -- i.e. computation.

So acquisition, possession, and use of knowledge involve computation. Consciousness and overt communication are not required. Computation (including the use of symbols or representations) is prior to consciousness and communication. It provides the framework within which a spectrum of increasingly sophisticated mechanisms can be explored, only a subset of which are conscious. (What this means is not as obvious as we may think. I have begun to relate consciousness to forms of computation in chapter ten of my book.)

In view of all this, any good theory of knowledge must give an account of the computational mechanisms and processes underlying observable behaviour.

2. Resources for knowledge

A shallow study of the knowledge of a system merely asks what knowledge it has, and perhaps which behaviours use which knowledge. Deeper investigations examine the underlying formal structures and the types of manipulations thereon, which make it possible to have and use that kind of knowledge.

Prior to any specific sort of knowledge are the general resources -- symbol generating mechanisms, symbol storing and manipulating mechanisms, information collection and transfer mechanisms (e.g. sensory

transducers). So a computational epistemology must include a study of forms of symbolism and methods of reasoning, interpreting, etc. As we have seen, logical symbolisms and processes are but one case, of limited use, though they are important, especially as we have a well developed theory of how they work.

Some computational resources may be quite general, others geared to the nature of the environment. For instance, a "cognitively friendly environment" (CFE) enables a trade-off between generality and efficiency or power. An environment is relatively cognitively friendly if the objects found in it cluster into relatively small subsets of the set of theoretically possible objects. This enables perceptual mechanisms to use redundancy in perceptual information, for instance jumping to conclusions on the basis of partial views. Wide-spread forms of cognitive friendliness allow genetically determined specialised mechanisms. For instance, in very many locations on earth there is a rich supply of electromagnetic radiation in the same range of wavelengths, a transparent atmosphere, and opaque, mostly rigid objects. Specialised computational properties of the visual system can evolve to take advantage of this. This is "compiled" knowledge. Such mechanisms are often mistaken for knowledge-free systems.

Variable forms of cognitive friendliness necessitate learning, and therefore more general computations. An open question: what initial resources are necessary for typically human forms of knowledge to develop in a few months or years in a CFE? This question cannot be answered by observation -- theoretical analysis and computational experiments are required, for a neonate, like a computer, need not reveal in its behaviour the most important internal computations.

3. Computational architecture

The afore-mentioned resources may be organised in an infinite variety of large-scale systems, composed of communicating subsystems. The computational architecture of the human mind is largely an open question. We can easily distinguish sub-functions e.g. perception (several varieties), self-monitoring, goal selection, plan-formation, plan-execution, inference, information storage, information recall, formation and modification of desires and attitudes etc, etc. How far do sub-functions correspond to sub-systems? E.g. can inference be separated from perception and plan-formation? Can storage of factual information be separated from the representation of motives? What are the trade-offs? Can the architecture of an individual change? What kinds of inter-system communication can occur -- e.g. are computer networks a good model? Do power hierarchies and resource-allocation play a role? (Mental politics and economics?)

Within specialist subsystems, the need to avoid combinatorial searches suggests a highly redundant architecture: large numbers of results of previous computations are stored instead of only basic principles and general inference mechanisms. For instance, instead of, or in addition to using a general purpose grammar, which enables us to generate or recognise an infinitely varied set of sentences, we may store many sentences, phrases, or sentence schemas which are compatible with the grammar, and make use of them directly instead of the more general mechanism, wherever this is possible. This puts a premium on powerful accessing mechanisms. Recognition can then substitute for reasoning. For many human abilities the constraints on processing include very rapid access to a very large store using incomplete or

possibly distorted "keys", like a partially distorted image, or a word partially obscured by background conversation.

Several studies in Artificial Intelligence point to the need for the same entity (plan, sentence, visual image) to be processed simultaneously at several levels of abstraction, with different structures at different levels. It is possible that there are forms of computation which are capable of achieving this that we have not yet begun to think of.

Another important constraint is the need to be able to construct very complex rapidly changing representations, e.g. in visual perception. Are list-processing and garbage-collection mechanisms used by organisms? Compare constant re-writing of an array-like structure.

4. Conditions for "meaningful" use of symbols

How can a machine attach meaning to symbols it manipulates? I have already indicated that it depends essentially on having a class of applicable procedures. The lowest level of "procedural" meaning found in any computer is essentially based on causal relations. The lowest level machine language is "understood" simply in terms of physical structures causing physical processes. "Descriptive" meaning and higher-level procedural meaning (e.g. goals expressed descriptively) can emerge from procedural capabilities in a manner which depends in part on the ability to interpret "conditional" instructions, in part on an appropriate computational architecture, providing an ability to check and correct mistaken information and a motivational subsystem which uses stored descriptions for achieving goals -- i.e. beliefs presuppose motives. It is not clear how far sensory detection procedures are required to define descriptive symbols, and how far abstract "axiomatic" systems suffice.

An intelligent system must be able to recognise when it is wrong and modify its procedures or store of information. Using the concept of correcting one's mistakes presupposes at least a primitive grasp of a concept of objective reality: a world beyond self. Any dynamic knowledgeable system capable of correcting its mistakes or learning about changes in the world must use representational resources whose potential extends beyond what is actually represented at any one time.

Generative power is thus one criterion for selecting between symbolisms. Generative power can be achieved in principled or ad hoc ways. The human mind seems to make heavy use of the latter -- trading conceptual economy for efficiency, and space for time. Another requirement is the need to be able to express generalisations, and to cope with varieties of incomplete information by means of symbolisations of varying specificity. This is where Fregean representations score heavily over analogical ones.

6. Conclusion

Investigation of these questions of "computational epistemology" provides a theoretical framework for philosophical and psychological studies of knowledge and cognitive processes, including what McCarthy calls "meta-epistemology" e.g. doing thought experiments involving simplified agents in simplified worlds, to discover the powers and limitations of various representational systems and strategies. Existing formal systems (e.g. logic, formal grammars) may turn out to be mathematical abstractions not closely related to what goes on in minds or brains, and incapable of supporting the computations required "in

real life[11], where constraints of time in particular can rule out otherwise mathematically adequate procedures. Equally, all the forms of computer programs which we now use in designing intelligent systems might turn out to be grossly inadequate to the task of accounting for the full power of human and animal intelligence. My own guess is that a subset of known forms of computation will be theoretically important for a long time. But it may not include what we now find easiest to program: manipulations of numbers.

References

Boden, Margaret Artificial Intelligence and Natural Man, Harvester Press and Basic Books, 1977.

Sloman, Aaron The Computer Revolution jm Philosophy; Philosophy Science and Models of Mind, Harvester Press and Humanities Press, 1975

Sussman, 6.J. A Computational Model of Skill Acquisition,

Winston, Patrick, Artificial Intelligence, Addison Wesley 1977.

Winston, Patrick (ed). The Psychology of Computer Vision, McGraw Hill, 1975.