ON LEARNING ABOUT NUMBERS
(Some problems and speculations.)

By Aaron Sloman, School of Social Sciences, University of Sussex.

Abstract

The aim of this paper is methodological and tutorial. It uses elementary number competence to show how reflection on the fine structure of familiar human abilities generates requirements exposing the inadequacy of initially plausible explanations. We have to learn how to organise our common sense knowledge and make it explicit, and we don't need experimental data so much as we need to extend our model-building know-how.

oooooooooo

Introduction

Work in A.I. needs to be informed by accurate analysis of real human abilities if it is to avoid exaggerated claims, and excessive concern with toy projects. The reflective method advocated here has much in common with the approach of some linguists and with philosophical analysis of things we all know, as practised by Frege, Ryle, Austin, Wittgenstein and others. Philosophers' analyses are distorted by their preoccupation with old puzzles and paradoxes, and by their failure to think about the problems of designing symbol-manipulating (information processing) mechanisms. Psychologists, with a few exceptions (e.g. Piaget, Wertheimer, Heider) miss out on the analysis altogether, partly because they confuse it with introspection, partly because they are driven by the myth that to be a scientist is to collect new data, and partly because the technique is hard to learn and teach.

The analysis of elementary number competence, given below, is mixed up with speculation about mechanisms. A metaphor now taken for granted, though perhaps one day it will have to be abandoned, is that acquiring and using knowledge requires a memory containing vast numbers of "locations" at which symbols of some kind can be stored. They need not be spatial locations, since points in any symbolic space will do, such as frequencies of radio waves, or structures of molecules. So my remarks below about locations and addresses which identify them make no assumptions about the medium used, except that it provides enough locations at which symbols can be stored, including symbols which identify locations in memory, i.e. "pointers". I make no assumptions about the mechanisms making addressing possible except that explicit addressing takes a negligible amount of time. It makes no difference for present purposes whether the locations are brain cells, molecules, frequencies of brain waves, or parts of some spiritual mechanism. Physiology is irrelevant to many problems about the structures and functions of mental mechanisms.

The main problem to be discussed here is: What is elementary number competence and how is it possible? The first task is to make explicit our common sense knowledge about what sorts of things are possible. (Not laws of behaviour, but possibilities are what we first need to explain. There are very few laws of human behaviour, but very many possibilities.) By thinking about the mechanisms required to explain these possibilities we begin to reveal the poverty of most philosophical and psychological theories about the nature of mathematical concepts and knowledge: they hardly begin to get to grips with the details we all know.

Number concepts aren't simple things you either get or don't get, but complex extendable structures built up gradually. Reflecting on even the

simplest things we know children can learn, shows that children somehow cope with quite complex computational problems. Some of these problems are common to many forms of learning, others peculiar to numbers and counting. For any small subset of the problems, any competent programmer could suggest several possible explanatory mechanisms. The difficulty lies in understanding what sorts of mechanisms might not only solve a few specific problems, but could form part of a larger mechanism explaining much more. There is a serious need to extend our knowledge of varieties of possible computational mechanisms.

The particular problems to be discussed here are concerned with knowing number words, knowing action sequences (like counting), and enriching one's understanding of a previously learnt sequence. Many more questions will be asked than answered.

Knowing number words

A child learns to recognise sounds like "one", "two", "number" and "count". An untutored view is that repeated exposure causes the sound to be stored, so that new occurrences can be recognised by matching. Immediately all sorts of questions can be asked. In what form is the sound represented - is it analysed into recognisable fragments, such as phonemes? How are experiences selected as worth storing? How is a matching item found in the vast store of memories when a word is recognised? Is an index used for finding items, and if so how does a child know about index construction? How is the matching between perceived and stored items done? Are variations coped with by storing variant forms or by using a flexible matching procedure or both? In the first case, how is the equivalence of stored variants represented? Why is repeated hearing sometimes needed for learning - is it because the child needs to experiment with different modes of analysis, representation and matching, in order to find a good way of dealing with variations? If so, how are the experiments managed? Why is repetition sometimes not needed for learning? When a new word is learnt how is new storage space allocated? How is the ability to say the word represented? Is output controlled by the same representation as recognition? How are different output styles associated with the same item, such as English and French number names, Arabic and Roman numerals? Does being able to count in different languages/explicit storage of different sequences, or is the same sequence used with a decision about output style at each step? Or can both methods be used?

Using only 26 letters we can construct thousands of words. A frequently used principle of computation is that if a small set of symbols is available and quickly recognisable (e.g. because the set is small and the matching simple), then a very much larger efficiently usable set of symbols can be made available, each consisting of some combination of symbols from the small set. By imposing an arbitrary order on the original set of symbols, we can make processes of storing and retrieving large numbers of the new symbols look like fast parallel searches, for instance in the way we use alphabetical order to find a name in a directory without exhaustive search. Alternatively, recognition of a complex item may take the form of computing a description, using recognition of the components, as in parsing a sentence or finding the average of a set of numbers (constructive recognition). So perhaps analysis of words into syllables, phonemes, or other sub-structures is used by children to facilitate storage and recognition of the thousands of words they learn. This attributes to toddlers sophisticated but unconscious computational abilities (e.g. the construction and use of indexes, decision trees, parsers). What do we know about possible mechanisms?

It is often suggested that some of the remarkable efficiency of human memory could be explained by a content-addressable store, i.e. a large collection of storage units each capable of comparing its contents with a broadcast pattern, and shouting "Here it is" to a central processor. However, this leaves problems about explaining our ability to cope with items varying enormously in size and complexity, such as letters, words, phrases, sentences, poems, plays, the number sequence, etc., and our ability to retrieve on the basis of elaborate inferences rather than simple matches: e.g. "What's the smallest three-digit number which rhymes with 'heaven' and contains a repeated digit?". The central processor would need to be able to transform questions into forms likely to produce responses from relevant storage units. This requires some kind of index or catalogue of the contents of those units, which would make their content-addressability redundant! Most of this paper is concerned with problems of indexing.

Associations between learnt items

Merely being able to tell whether an item has been met before is not of much use. More must be known about it: such as how to produce it, in what forms it may be experienced, that it is a word, that it belongs to a certain syntactic class, that it has certain uses, that it is one of a group of words with related meanings or uses (a semantic field), that various objects and procedures are associated with it, and so on. Associationist psychologists and empiricist philosophers are obviously right in claiming that much knowledge depends on learnt associations. But they have been so concerned with the external conditions for establishing such associations that they have hardly begun to think about the problems of how such knowledge might be represented, stored and manipulated so as to be accessible, usable, and if necessary modifiable. (Explanations which convince one's colleagues are sometimes seen to be inadequate only as a result of attempting to design a mechanism actually able to do these things.)

Any one item may have to be associated with very many others. The word "word" is somehow linked to thousands of instances, and the item representing one's home town linked to very many facts known about that town. Similarly, we expect children to pick up many facts about an individual number, such as that it is a number, that it is used in counting, what its successor is, what its predecessor is, whether it is odd or even, whether it is prime and if not then what its factors are, which pairs of numbers add up to it, the result of adding or multiplying it with various others, how to say it, how to write it, how to recognise it when said or written in various styles, how to bypass counting by recognising spatial patterns corresponding to it, what it can be used for, how to count forwards from it, how to count backwards from it, where it lies in relation to various "landmarks" in the number sequence, and so on. (See figure 1.) Why should we expect children to pick up so many associations? The process of building up those associations is a long one and involves many mistakes which get corrected. An explanatory theory must specify a mechanism which is not merely able to hold the finished structure in an efficiently accessible form, but is also capable of explaining how such structures can be built up, how they are modified, how they are used, etc. I do not believe educational psychologists have the foggiest notion of what such a mechanism might be like. Yet gifted teachers have some intuitive grasp of how it works.

Take the question "What's after three?". The problem is not merely to find something associated with "three" and "after". Besides "four", "two" will be associated with them, and so may lots of pairs of numbers be, e.g. pairs N and K for which it is known that N is K after three: five is

two after three. So getting to the required association requires the ability to analyse the question (which may be ambiguous) and use the analysis to control the search for relevant links in the store of associations. (E.g. in figure 1, find the node representing three then search for a link from it labelled "successor". Do children learn to translate the original question into this kind of internal procedure? How?)

There are many ways in which associations can be stored, and different structures require different procedures for their use. A common method in computing is to use "property-lists" or "association-lists", as in figure 2, which shows a chain of links where each link contains two storage cells treated as an association by the memory mechanism. A chain may be attached to some item, e.g. to the concept "numbers", and related items are "hung" from the chain by means of pointers giving their addresses. As figure 3 shows, the items hung from the chain may themselves be associations, corresponding to the labelled links of figure 1. Thus in the context of the chain attached to "three", there is an association between "predecessor" and "two", whereas in a chain attached to "four" (not shown) there would be an association between "predecessor" and "three". Associations are relative to context.

Stored structures are not enough. Procedures are required for creating and finding associations in them. For instance, the following procedure will generate a search down a chain starting at LINK, looking for an association of type LABEL, in a structure like figure 3, and will return the associated item as its result.

```
PROCEDURE FINDASSOC (LINK,LABEL);
   WHILE ( HD(HD(LINK)) ≠ LABEL ) REPEAT
     (ASSIGN  TL(LINK) TO LINK);
   RETURN(TL(HD(LINK)));
END
```

So FINDASSOC(THREE, TYPE), yields a pointer to NUMBER as its result, in figure 3. A more complex procedure is required for adding a new association: it will have to get a free link (how?) and insert it at a suitable place in the chain, with its HD pointing to the new association and its TL pointing to the next link in the chain, if any. If children do anything like this to store and use associations, then how do they build up such chains, and how do they come to know the procedures for finding required associations? Are these inborn mechanisms? Clearly not all procedures for getting at stored information are innate. For instance, children have to learn how to count backwards or answer "What's before 'four'?" even though they may already know the order of the numbers. More on this later.

Learning a sequence

In this paper I shall not consider the more advanced stage where a child grasps a rule for generating indefinitely many number names, e.g. using decimal notation. An earlier stage involves learning to recognise not only isolated words, but also a sequence "one", "two", "three", etc. This is common to many things children learn. Some learnt sequences are made up of already meaningful parts which combine (how?) to form a new meaningful whole, like "Mary had a little lamb...", whereas other sequences, like the alphabet and numbers used in early counting games, are arbitrary, when first learnt. Sequences with varying amounts of significant structure include: the days of the week, the letters used to spell a word, the sounds in a spoken word, the sequence of intervals in a song, the steps required to assemble a toy, routes frequently travelled, recipes, and various games and rituals. An adequate explanation of how the simple and arbitrary

sequences are learnt, or stored or produced should also be part of an explanation of the ability to cope with more complex structures containing simple sequences as parts, such as nursery rhymes which have many levels of structure, and action procedures which, besides simple sequences, also contain loops, conditional branches, sub-procedures, gaps to be filled by decision-making at execution time, and other forms of organisation.

All this points to the old idea (compare Miller, et al.) that human abilities have much in common with computer programs. But further reflection on familiar facts shows that programs in the most common programming languages don't provide a rich enough basis for turning this from a thin metaphor into an explanatory theory. For instance, people can excute unrelated actions in parallel. Moreover, they apparently don't require their procedures to have built-in tests to ensure that conditions for their operation continue to be satisfied, with explicit instructions about what to do otherwise, like instructions for dealing with the end of a list. All sorts of unpredictable things can halt a human action at any stage (like learning one's house is on fire) and a decision about what to do can be taken when the interruption occurs, even if no explicit provision for such a possibility is built into the plan or procedure being executed. These points suggest that models of human competence will have to use mechanisms similar to operating systems for multi-programmed computers. For instance, an operating system can run a program, then interrupt it when some event occurs even if the program makes no provision for interruption. Similarly, if something goes wrong with the running of the program, like an attempt to go beyond the end of a list, the program breaks down, but the operating system or interpreter which runs the program can decide what to so, e.g. send a message to the programmer, so that there is not a total breakdown. Of course, the operating system is just another program. So the point is simply that to make the program metaphor fit human abilities, we must allow not merely that one program can use another as a subroutine, but that some programs can execute others and control their execution, in a parallel rather than a hierarchic fashion. (These arguments are familiar to many people in A.I.)

In counting objects, a child has to be able to generate different action sequences in parallel, keeping them in phase. Thus the process of saying number names, controlled by an internal structure, and the process of pointing in turn at objects in some group, controlled by the external structure, have to be kept in phase. In a suitable programming language one could keep two processes in phase by means of a procedure something like

```
PROCEDURE COEXECUTE (PROCESS1, PROCESS2, STOPPING-CONDITION);
 START:
  STEP(PROCESS1);
  STEP(PROCESS2);
  IF NOT(STOPPING-CONDITION) THEN GOTO START;
END
```

Unfortunately, this is not an acceptable model in view of the familiar fact that children (and adults doing things in parallel) sometimes get out of phase when counting and (sometimes) stop and correct themselves. This suggests that keeping the two sequences in phase is done by a third process something like an operating system which starts the processes at specified speeds, but monitors their performance and modifies the speeds if necessary, interrupting and perhaps restarting if the sequences get out of phase, which would be impossible with the procedure COEXECUTE. It is as if we could write programs something like:

```
PROCEDURE RUNINPHASE(PROCESS1, PROCESS2);
DO IN PARALLEL (a) to (d):
   (a) RUN PROCESS1;
   (b) RUN PROCESS2;
   (c) IF PROCESS1 AND PROCESS2 BEGIN TO GET OUT OF PHASE THEN
         MODIFY SPEED OF PROCESS1 OR PROCESS2 TO KEEP IN PHASE;
   (d) IF PROCESS1 AND PROCESS2 GET OUT OF PHASE THEN RESTART THEM;
END
```

The computational facilities required for this kind of thing are much more sophisticated than in COEXECUTE and are not provided in familiar programming languages. (Monitoring interactions between asynchronous parallel processes may be an important source of accidental discoveries (creativity) in children and adults.)

Further, the child has to be able to apply different stopping conditions for this complex parallel process, depending on what the task is. So it should be possible for yet another process to run the procedure RUNINPHASE, watching out for appropriate stopping conditions. For instance, when the question is "How many buttons are there?" use "No more buttons" as main stopping condition, whereas in response to a request "Give me five buttons", use "Number five reached" as main stopping condition. I say main stopping condition, because other conditions may force a halt, such as getting out of phase or running out of numbers or (in the second case) running out of buttons. How do children learn to apply the same process with different stopping conditions for different purposes? How is the intended stopping condition plugged into the process? This would be trivial for a programmer using a high-level language in which a procedure (to test for the stopping condition) can be given as a parameter to another procedure - but do children have such facilities, or do they use mechanisms more like the parallel processes with interrupt facilities described here? These parallel mechanisms might also explain the ability to learn to watch out for new kinds of errors. E. g. having learnt to count stairs where there is no possibility of counting an item twice, learning to count buttons or dots requires learning to monitor for repetition and omission. There are many ways this could be organised.

If we consider what happens when a child learns to count beyond twenty, we find that a different kind of co-ordination between two sequences is required, namely the sequence "one, two, three ... nine" and the sequence "twenty, thirty, ... ninety". Each time one gets round to "nine" in the first sequence one has to find one's place in the second sequence so as to locate the next item. A programmer would find this trivial, but how does a child create this kind of interleaving in his mind? And why is there sometimes difficulty over keeping track of position in the second sequence "... fifty eight, fifty nine, ... um .. er, thirty, thirty one..."? Clearly this is not a problem unique to children: we all have trouble at times with this sort of book keeping. But how is it done when successful? And what kind of mechanism could be successful sometimes yet unsuccessful at others? My guess is that human fallibility has nothing to do with differences between brains and computers as is often supposed, but is a direct consequence of the sheer complexity and flexibility of human abilities and knowledge, so that for example there are always too many plausible but false trails to follow. When computers are programmed to know so much they will be just as fallible, and they'll have to improve themselves by the same painful and playful processes we use.

We have noted a number of familiar aspects of counting and other actions which suggest that compiled programs in commonly used programming

languages don't provide a good model for human abilities. A further point to notice is that we not only execute our procedures or programs, we also build them up in a piecemeal fashion (as in learning to count), modify them when they seem inadequate, and examine them in order to anticipate their effects without execution. We can decide that old procedures may be relevant to new problems, we can select subsections for use in isolation from the rest, and we may even learn to run them backwards (like learning to count backwards). This requires that besides having names and sets of instructions, procedures need to be associated with specifications of what they are for, the conditions under which they work, information about likely side-effects, etc. The child must build up a catalogue of his own resources. Further, the instructions need to be stored in a form which is accessible not only for execution but also for analysis and modification, like inserting new steps, deleting old ones, or perhaps modifying the order of the steps. Such examination and editing cannot be done to programs as they are usually stored.

List structures in which the order of instructions is represented by labelled links rather than implicitly by position in memory would provide a form of representation meeting some of these requirements (and are already used in some programming languages). Thus, figure 2 can be thought of either as a structure storing information about number names (an analogical representation of their order), or else as a program for counting. The distinction between data structures and programs has to be rejected in a system which can treat program steps as objects which are related to one another and can be changed. We explore some consequences of this using counting as an example.

Learning to treat numbers as objects with relationships

There are several ways in which understanding of a familiar action sequence may be deficient, and may improve. One may know a sequence very well, like a poem, telephone number, the spelling of a word, or the alphabet, yet have trouble reciting it backwards. One may find it hard to start from an arbitrary position in a sequence one knows well, like saying what comes after "K" in the alphabet, or starting a piano piece in the middle. But performance can improve. A child who counts well may be unable easily to answer "What comes after five?". Later, he may be able to answer that question, but fail on "What comes before six?", "Does eight come earlier or later than five?" and "Is three between five and eight?". He doesn't know his way about the number sequence in his head, though he knows the sequence. Further, he may understand the questions well enough to answer when the numbers have been written down before him, or can be seen on a clock. (There are problems about how this is learnt, but I'll not go into them.) Later, the child may learn to answer such questions in his head, and even to count backwards quickly from any position in the sequence he has memorised. How? To say the child "internalises" his external actions is merely to label the problem: moving back and forth along a chain of stored associations is quite a different matter from moving up and down staircases or moving one's eye or finger back and forth along a row of objects.

There are at least two kinds of development of knowledge about a stored structure (which may be a program), namely learning new procedures for doing things with the structure, and extending the structure so as to contain more explicit information about itself. The former is perhaps the more fundamental kind of development of understanding, while the latter is concerned with increased facility. A very simple procedure enables a chain like that in figure 2 to be used to generate a sequence of actions, for example:

```
PROCEDURE SEQUENCE (LINK);
 START:
  OUTPUT(HD(LINK));
  ASSIGN TL(LINK) TO LINK;
 GOTO START;
END
```

or

```
PROCEDURE SEQUENCE (LINK);
 OUTPUT(HD(LINK));
 SEQUENCE(TL(LINK));
END
```

Going down the chain starting from a given link is thus easy, and a procedure to find the successor of an item would use a similar principle. But answering "What's before item X?" is more sophisticated, since on getting to a particular location (e.g. the link whose HD points to X), one does not find there any information about how one got there, so the last item found must be stored temporarily. One method is illustrated in the following procedure.

```
PROCEDURE PREDECESSOR (X,LINK);
 LOCAL VARIABLE TEMP; ASSIGN "NONE" TO TEMP;
  START:
   IF HD(LINK)=X THEN RETURN(TEMP)
    ELSE ASSIGN HD(LINK) TO TEMP AND ASSIGN TL(LINK) TO LINK AND
    GOTO START;
END
```

How could a child learn to create a procedure like this or the more elegant versions a programmer would write? Does he start with something more specialised then somehow design a general method which will work on arbitrary chains? Perhaps it has something to do with manipulating rows of objects and other sequences outside one's head, but to say this does not give an explanation, since we don't know what mechanisms enable children to cope with external sequences, and in any case, as already remarked, chains of associations have quite different properties. For a child to see the analogy would require very powerful abilities to do abstract reasoning. Maybe the child needs them anyway, in order to learn anything.

In any case, merely being able to invent procedures like PREDECESSOR is not good enough. For some purposes, such as counting backwards quickly, we want to be able to find the predecessor or successor of an item much more quickly than by searching down the chain of links until the item is found. If a child knew only the first four numbers, then he could memorise them in both directions, building up the structure of figure 4 instead of figure 2. Notice that this use of two chains increases the complexity of tasks like "Say the numbers", or "What's after three?", since the right chain has to be found, while reducing the complexity of tasks like "Say the numbers backwards," and "What's before three?" However, when a longer sequence had been learnt, this method would still leave the need to search down one or other chain to find the number N in order to respond to "What's after N?", "What's before N?", "Count from N", "Count backwards from N", "Which numbers are between N and M?", etc., for there is only one route into each chain, leading to the beginning of the chain. For instance, when one has found the link labelled X (figure 4) one knows how to get to the stored representation of "three", but it is not possible simply to start from the representation of "three" to get to the links which point to it in the two chains. So we need to be able to associate with "three" itself information about where it is in the sequence, what its predecessor is, what its successor is, and so on.

A step in this direction is shown in figure 5, where each number name is associated with a link which contains addresses of both the predecessor and the successor, like the link marked V, associated with "two". The information that the predecessor is found in the HD and the successor found

in the TL would be implicit in procedures used for answering questions. However, if one needed to associate much more information with each item, and did not want to be committed to having the associations permanently in a particular order, then it would be necessary to label them explicitly, using structures like those in figure 1 and figure 3, accessed by a general procedure like FINDASSOC, defined previously.

To cut a long story short, the result of explicitly storing lots of discoveries about each number, might be something like figure 6, which is highly redundant. The structure may look very complex, yet using it to answer questions requires simpler procedures than using, say figure 2, for, having found the link representing a number, one can then find information associated with that number by simply following forward pointers from it, e.g. using FINDASSOC, whereas in figure 2 or 5 finding the predecessor and successor of a number requires using two different procedures, and each requires a search down a chain of all the numbers to start with. Of course, a structure like figure 6 provides simple and speedy access at the cost of using up much more storage space. But in the human mind space does not seem to be in short supply!

If an item in a structure like figure 6 has a very long chain of associations, it might be preferable to replace the linear chain with a local index to avoid long searches. This would require the procedure FINDASSOC to be replaced by something more complex. Alternatively, one could easily bring a link to the front of the chain each time the association hanging from it is used: this would ensure that most recently and most frequently used information was found first, without the help of probabilistic mechanisms.

Notice that in a structure like this, normal "part-whole" constraints are violated: information about numbers is part of information about "three", and vice versa. So by using pointers (addresses) we can allow structures to share each other. In a rich conceptual system circular definitions will abound. If knowledge is non-hierarchic, as this suggests, then perhaps cumulative educational procedures are quite misguided. Further, this kind of structure does not need a separate index or catalogue specifying where to look for associations involving known items, for it acts as an index to itself, provided there are some ways of getting quickly from outside the structure to key nodes, like the cells containing "three" and "number". (This might use an index, or content addressable store, or indexing tricks analogous to hash coding, for speedy access.) The use of structures built up from linked cells and pointers like this has a number of additional interesting features, only a few of which can be mentioned here. Items can be added, deleted, or rearranged merely by changing a few addresses, without any need for advance reservation of large blocks of memory or massive shuffling around of information, as would be required if items were stored in blocks of adjacent locations. The same items can occur in different orders in different structures which share information (see figure 4 for a simple example). Moreover, the order can be changed in one sequence without affecting another which shares structure with it. For instance, in figure 4 the addresses in links W, X, Y, and Z can be changed so as to alter the order of numbers in chain labelled "reverse" without altering the chain labelled "forward".

As we saw in connection with figure 2, when the rest of the mechanism is taken for granted, a structure of the kind here discussed looks like a program for generating behaviour, but when one looks into problems of how the structure gets assembled and modified, how parts are accessed, how different stopping conditions are applied, etc., then it looks more like a

data structure used by other programs. If the distinction between programs and data structures evaporates, then don't some A.I. slogans about procedural knowledge have to be retracted, or at least clarified? (Compare Hewitt 1971.).

Conclusion

Further simple-minded reflection on facts we all know reveals many gaps in the kinds of mechanisms described here. For instance, very little has been said about the procedures required for building, checking, modifying, and using a structure like figure 6. Nothing has been said about the problems of perception and conception connected with the fact that counting is not applied simply to bits of the world but bits of the world individuated according to a concept (one family, five people, millions of cells - but the same bit of the world counted in different ways). Nothing has been said about recognition of numbers without explicit counting. Nothing has been said about how the child discovers general and non-contingent facts about counting, such as that the order in which objects are counted does not matter, rearranging the objects does not matter, the addition or removal of an object must change the result of counting, and so on. (Philosophers' discussions of such non-empirical learning are so vague and abstract as to beg most of the questions.) I cannot explain these and many more things that even primary school children learn. I don't believe that anybody has even the beginnings of explanations: only new jargon for labelling the phenomena.

I have offered all this only as a tiny sample of the kind of exploration needed for developing our abilities to build theoretical models worth taking seriously. In the process our concept of mechanism will be extended and the superficiality of current problems, theories and experiments in psychology and educational technology will become apparent.

Philosophers have much to learn from this sort of exercise too, concerning old debates about the nature of mind, the nature of concepts and knowledge, varieties of inference, etc. Consider answers they have given to the question "What are numbers?", namely: numbers are non-physical mind-independent entities (Platonists), numbers are perceivable properties of groups of objects (Aristotle?), numbers are mental constructions whose properties are found by performing mental experiments (Kant and Intuitionists), numbers are sets of sets, definable in purely logical terms (Logicists), numbers are meaningless symbols manipulated according to arbitrary rules (Formalists), they are whatever satisfy Peano's axioms (Mathematicians) or numbers are simply a motley of things which enter into a variety of "language" games" played by different people (Wittgenstein). (These descriptions are too brief for accuracy or clarity; for more detail consult books on philosophy of mathematics, e.g. Korner's.) Further work will show that each of these views is right in some ways, misleading in others, but that none of them gets near an accurate description of all the rich structure in our number concepts.

I believe the old nature-nurture (heredity-environment) controversy gets transformed by this sort of enquiry. The abilities required in order to make possible the kind of learning described here, for instance the ability to construct and manipulate stored symbols, build complex networks, use them to solve problems, analyse them to discover errors, modify them, etc., - all these abilities are more complex and impressive than what is actually learnt about numbers! Where do these abilities come from? Could they conceivably be learnt during infancy without presupposing equally powerful symbolic abilities to make the learning possible? Maybe the much

discussed ability to learn the grammar of natural languages (cf. Chomsky) is simply a special application of this more general ability? This question cannot be discussed usefully in our present ignorance about possible learning mechanisms.

Finally a question for educationalists. What would be the impact on primary schools if intending teachers were exposed to these problems and given some experience of trying to build and use models like figure 6 on a computer?

Acknowledgements

Some of these ideas were developed during tenure of a visiting fellowship in the Department of Computational Logic, Edinburgh. I am grateful to Bernard Meltzer and the Science Research Council for making this possible, and to colleagues in Edinburgh and at Sussex University for helping to remove the mysteries from computing. Alan Mackworth's criticisms of an earlier draft led to several improvements.

Bibliography

Austin, J. L. 'A plea for excuses', in his Philosophical Papers Oxford University Press, 1961, also in Philosophy of Action ed. by A. R. White, Oxford University Press, 1968.

Chomsky, N. Aspects of the Theory of Syntax, chapter 1. M.I.T. Press, 1965.

Frege, G. Philosophical Writings, translated by P. T. Geach and M. Black. Blackwell

Hewitt, C. 'Procedural embedding of knowledge in PLANNER', in Proc. 2nd IJCAI, British Computer Society, 1971.

Korner, S. Philosophy of Mathematics, Hutchinson, 1960.

Lindsay, P. H. and Norman, D. A. Human Information Processing, Academic Press 1972. Chapters 10 and 11 give a useful introduction to semantic networks.

Miller, G. A., Galanter, E. Pribram, K. M. Plans and the Structure of Behaviour, Holt Rinehart and Winston 1960.

Minsky, M. 'Form and Content in Computer Science', part 3, J.A.C.M. 1970.

Quillian, M. R. 'Semantic Memory' in Semantic Information Processing, ed. by M. Minsky, M.I.T. Press, 1968.

Ryle, G. The Concept of Mind, Hutchinson 1949. also Penguin Books.

Winston, P. Learning Structural Descriptions from Examples M.I.T. A.I. Lab. AI TR-321, 1970.

Wittgenstein, L. Philosophical Investigations, Blackwell, 1953. Remarks on the Foundations of Mathematics, Blackwell, 1956.

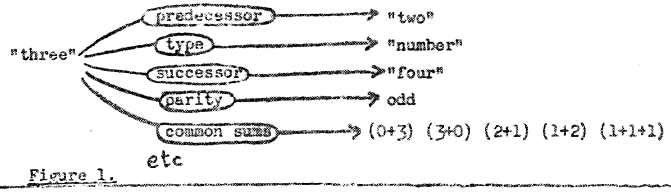
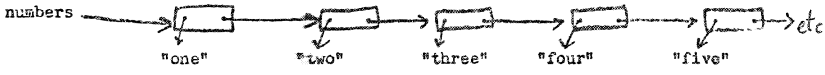


Figure 1.



The use of chains. Each link contains two cells, the first called HD the second TL, containing symbols which may be pointers to other locations. A cell points to a location in memory by containing an address of that location.

Figure 2.

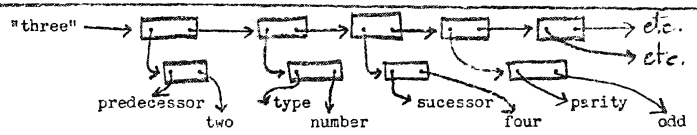


Figure 3. (This shows a translation of figure 1 using list structures.)

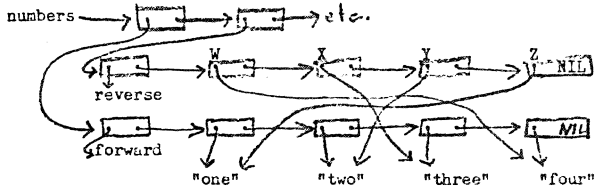
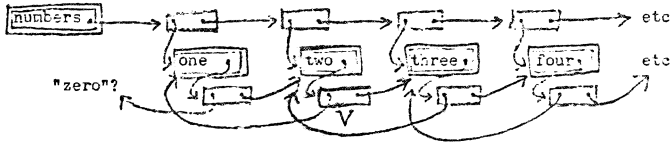


Figure 4. (The ability to count backwards from four.)



Here, writing a symbol inside a link represents the fact that that link may be addressed by means of the symbol, e.g. using an index, and the HD of the link contains a pointer back to the symbol.

Figure 5.

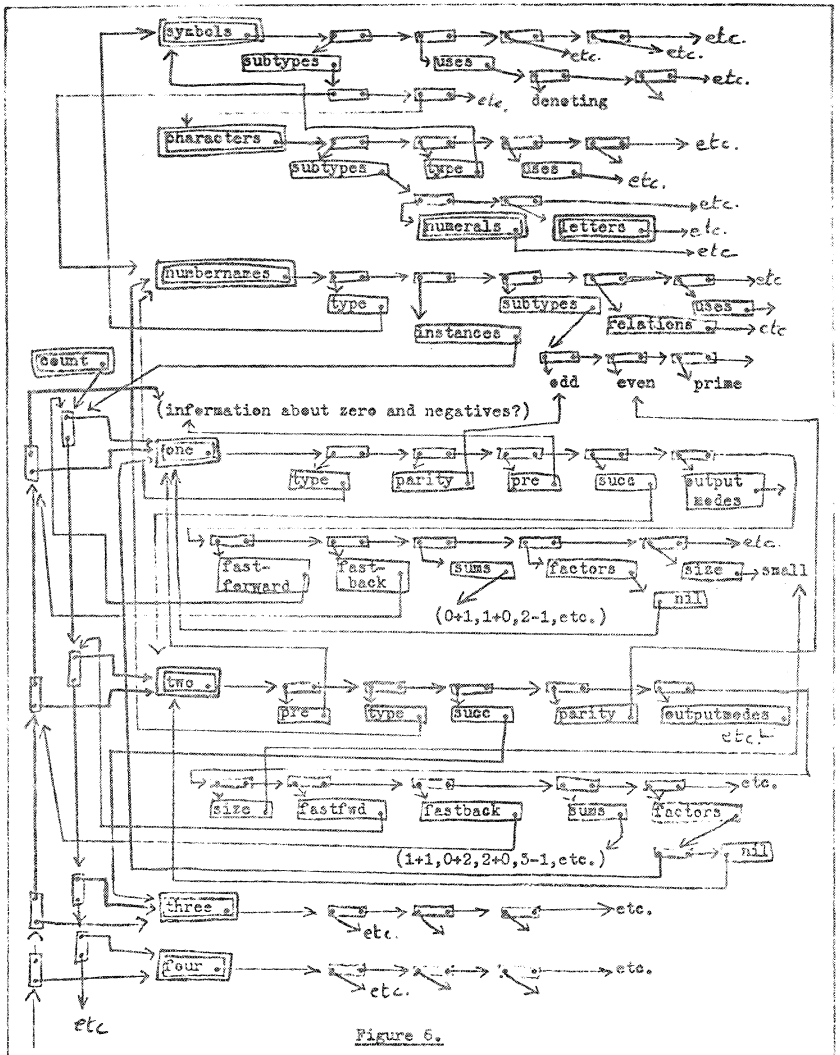


Figure 6.

This represents, with many oversimplifications and omissions, some of the information about the number sequence learnt by children. To save space and improve intelligibility, relations between numbers, number-names, and numerals (e.g. two, "two" and "2") have not been represented. Doubly-bored nodes are meant to be directly addressible from outside the network (compare figure 5). The two vertical chains on the left represent the ability to count forwards or backwards at speed, starting at any known number.