On the Analysis of Human Problem Solving Protocols

Allen Newell

June 27, 1966

This paper is to be given at the International Symposium on Mathematical and Computational Methods in the Social Sciences, Rome, July 4-9, 1966. It may not be reproduced without the permission of the author. This research was supported by Research Grant MH-07722-02 from the National Institutes of Health.

Carnegie Institute of Technology
Pittsburgh, Pennsylvania

June 27, 1966

On the Analysis of Human Problem Solving Protocols*

Allen Newell

The last decade has seen the emergence of information processing theories of human problem solving, expressed usually as computer programs that simulate behavior. These theories have led to a resurgence of interest in protocols as a source of data. The term protocol generally refers to a record of the time sequence of events. In the present context it includes also the continuous verbal behavior of the subject operating under instructions to "think aloud." Protocols match well some of the strong points of information processing theories, but also have several weak features. The purpose of this paper is to discuss the analysis of protocol data and to suggest one new line of attack for strengthening it. The intent is somewhat methodological, but some new material will be introduced.

The use of protocols is not at all new. Their connection with the introspective method, especially of the Wurzburgers [15], goes back to the first decades of this century. They served Duncker well in his classic contribution to the psychology of problem solving in 1935 [5]. They formed the primary material in the forties for an intensive study of thinking in chess by the Dutch psychologist DeGroot (recently revised and translated into English [4]). However, free verbal report fell into relative disuse within the mainstream of behavioristic psychology, especially in the United States. And not until the advent of the computer, with the corresponding conceptual development in programming, has it been possible to couple protocols with precise models of process.

* I am grateful to my colleague, H. A. Simon, both for his contribution to the substance of the work presented here and to his criticism of an earlier draft.

Let us start with a concrete example, which may be already familiar [24]. We divide the analysis into stages:

1. The subject, a college student, is given a problem in the elementary propositional calculus. This task is shown in Figure 1. He is instructed to say aloud whatever occurs to him throughout the problem.

2. The tape recording of his verbal behavior is transcribed and becomes the raw record of the experiment, along with a record of relevant non-verbal behaviors, such as writing down expressions. This is the <u>protocol</u>.

3. After intensive analysis (in terms of hours per minute of subject behavior) a proposal emerges for a scheme of information processing that will simulate the subject's behavior. This is shown in Figure 2; we may call it the <u>flow diagram</u>, although it can take varied forms.

4. A computer program (called GPS in this instance) is coded and debugged that outputs a record, called the <u>trace</u>, which purports to correspond to the behavior indicated in the protocol. Figure 3 shows a short sample of the protocol and trace, side by side.

The behavioral situation used in this example is one of deliberate, extended problem solving in a formal, abstract, symbolic task. Most attempts to work with this scheme of protocol analysis have involved tasks that can be similarly characterized. Since we will continue this focus in the present paper, let us note now that not all information processing theories deal with problem solving behavior [1, 3, 32, 33]. Nor is the only appropriate experimental paradigm for problem solving one involving simulation of individual protocols [8, 13, 14, 29]. Our narrowness of view here is conditioned primarily by the urge to fashion this one scheme of analysis into a more useful tool.

Objects are formed by building up expressions from letters (P, Q, R, ...) and connectives . (dot), v(wedge), $\supset$ (horseshoe), and - (tilde). Examples are P, -Q, PvQ, $-(R \supset S).-P$. --P is equivalent to P throughout.

Twelve rules exist for transforming expressions (where A, B, and C may be expressions or subexpressions):

R1. $A.B \rightarrow B.A$
$AvB \rightarrow BvA$

R2. $A \supset B \rightarrow -B \supset -A$

R3. $A.A \leftrightarrow A$
$AvA \leftrightarrow A$

R4. $A.(B.C) \leftrightarrow (A.B).C$
$Av(BvC) \leftrightarrow (AvB)vC$

R5. $AvB \leftrightarrow -(-A.-B)$

R6. $A \supset B \leftrightarrow -AvB$

R7. $A.(BvC) \leftrightarrow (A.B)v(A.C)$
$Av(B.C) \leftrightarrow (AvC).(AvC)$

R8. $A.B \rightarrow A$
$A.B \rightarrow B$
Applies to main expression only.

R9. $A \rightarrow AvX$
Applies to main expression only.

R10. $\left.\begin{matrix}A\\B\end{matrix}\right\} \rightarrow A.B$
A and B are two main expressions.

R11. $\left.\begin{matrix}A\\A \supset B\end{matrix}\right\} \rightarrow B$
A and $A \supset B$ are two main expressions.

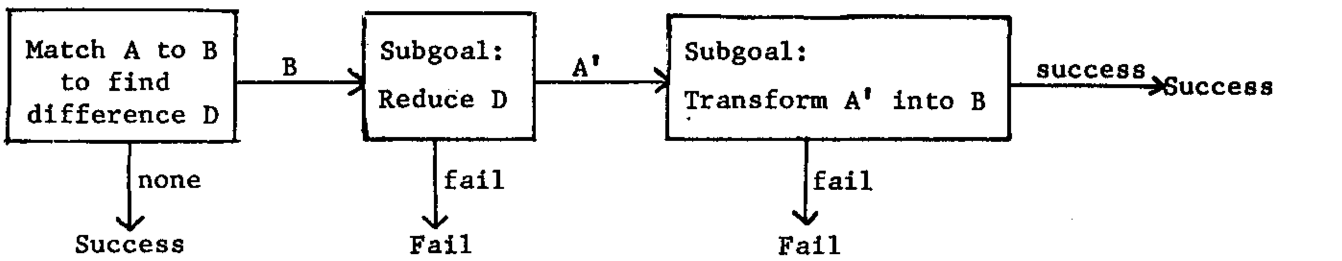
R12. $\left.\begin{matrix}A \supset B\\B \supset C\end{matrix}\right\} \rightarrow A \supset C$
$A \supset B$ and $B \supset C$ are two main expressions.

Example, showing subject's entire course of solution on problem:

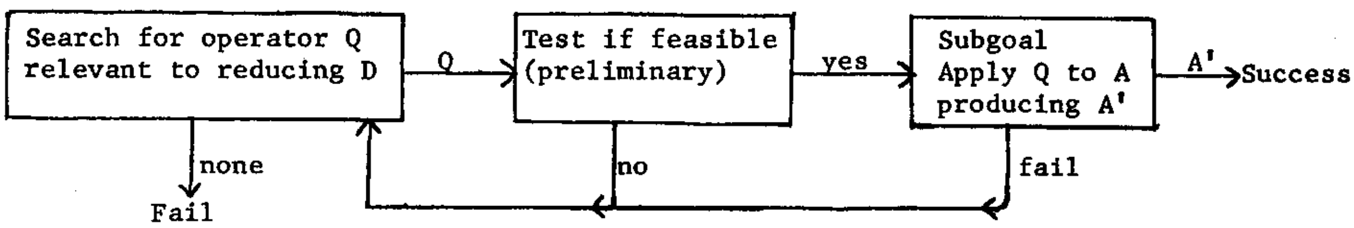
1. $(R \supset -P).(-R \supset Q)$	$-(-Q.P)$
2. $(-RvP).(RvQ)$	Rule 6 applied to left and right of 1.
3. $(-RvP).(-R \supset Q)$	Rule 6 applied to left of 1.
4. $R \supset -P$	Rule 8 applied to 1.
5. $-Rv-P$	Rule 6 applied to 4.
6. $-R \supset Q$	Rule 8 applied to 1.
7. RvQ	Rule 6 applied to 6.
8. $(-Rv-P).(RvQ)$	Rule 10 applied to 5 and 7.
9. $P \supset -R$	Rule 2 applied to 4.
10. $-Q \supset R$	Rule 2 applied to 6.
11. $P \supset Q$	Rule 12 applied to 9 and 6.
12. $-PvQ$	Rule 6 applied to 11.
13. $-(P.-Q)$	Rule 5 applied to 12.
14. $-(-Q.P)$	Rule 1 applied to 13. QED.

Figure 1: Logic Task.

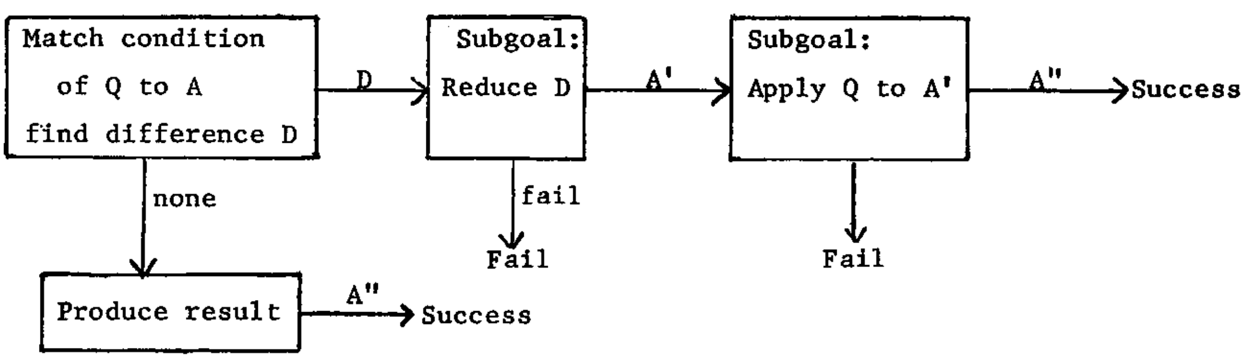
Goal: Transform object into object B



Goal: Reduce difference D between object A and object B



Goal: Apply operator Q to object A



For logic task of the text:

Feasibility test (preliminary):
Is the main connective the same? (e.g., A.B→B fails against PvQ)
Is the operator too big? (e.g., (AvB).(AvC) → Av(B.C) fails against P.Q)
Is the operator too easy? (e.g., A → A.A applies to anything)
Are the side conditions satisfied? (e.g., R8 applies only to main expressions)

Table of connections

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12
Add terms			x				x		x	x	x	x
Delete terms			x				x	x			x	x
Change connective					x	x	x					
Change sign					x							
Change lower sign		x			x	x						
Change grouping				x			x					
Change position	x	x										

x means some variant of the rule is relevant. GPS will pick the appropriate variant.

Figure 2: Flow diagram for GPS.

	Program trace	Protocol
B1	LO -(-Q.P) L1 (R⊃ -p).(-R⊃Q) GOAL 1 TRANSFORM L1 INTO LO	<no transcription of verbal behavior>
B2		Well, looking at the left hand side of the equation,
B3	GOAL 2 DELETE R FROM L1	first we want to eliminate one of the sides
B4	GOAL 3 APPLY R8 TO L1	by using rule number 8.
B5		It appears too complicated to work with first.
	PRODUCES L2 R⊃-P	
B6	GOAL 4 TRANSFORM L2 INTO LO	Now -
	GOAL 5 ADD Q TO L2	
	REJECT	
	GOAL 2	
	GOAL 6 APPLY R8 TO L1	
	PRODUCES L3 -R⊃Q	
	GOAL 7 TRANSFORM L3 INTO LO	
	GOAL 8 ADD P TO L3	
B7	REJECT	no, - no, I can't do that
B8		because I will be eliminating either the Q or the P in that total expression.

Figure 3: Initial segment of GPS simulation on S4 on problem D1.

The paradigm just presented has three dominant features. First, it deals with the dynamics of an individual episode of behavior. Second, it contains theoretical assertions about the behavior that are precise and highly specific. Third, it deals with the content of the task. Thus, the theory simulates behavior that is adequate to the task*. Involvement with content is also reflected in the use of freely produced linguistic utterances as the primary source of data. In this respect the protocol is a natural data form for this type of theory. It is appropriate, also, in providing a large amount of information per unit of time about the subject. The necessity for this becomes apparent upon considering how to identify a system as complex as a problem solving human.

The major problems in protocol analysis arise from these same dominant features. Let me mention two problems that are already prominent, before turning to a third that is my own greatest concern and the focus of this paper.

The problem of assessment. In assessing the validity of the program to describe or explain the subject's behavior, two things are missing to which psychologists have become accustomed. First, there is no acceptable way to quantify the degree of correspondence between the trace of the program and the protocol. This is not a problem of making the inference definite or public. Trace and protocol can be laid side by side, as is done in Figure 3. However, comparison still must be made between an elaborate output statement and a free linguistic utterance. Although a human can assess each instance qualitatively, there are no available techniques for quantifying the comparison, or summarizing the results of a large set of comparisons.

---

* More precisely, a simple and completely specified interpreter is sufficient to translate the statement of the theory into adequate behavior.

Second, the program has been created partly with the subject's protocol in view. Thus, something analogous to the calculation of degrees of freedom used in fitting curves with free parameters to data is appropriate. But programs are not parameterized in any simple way and no analytic framework yet exists for allowing for degrees of freedom.

The problem of theory. Programs are symbolic structures that specify the behavior through time of a system in the same manner as difference equations [25] -- thus, their availability as theories. Still a certain discomfort exists in the idea of programs as theories [30, chapter 2]. The discomfort stems partly from the specificity of the theory, since it appears to be limited not only to a single person, but to a single episode. While in other areas, such as the earth's geological development, we are content to construct a theory of the history of an individual system, it is clear that little scientific interest per se attaches to the particular college student of our example. Thus, we must view these individualized theories -- microtheories would be an appropriate term* -- as the way to bring a more general theory into contact with its data. This more general theory, of course, is neither so formalized nor precise as the microtheories to which it gives rise. In part it includes the basic possibility of viewing a human in a precise way as an information processing system. But it also includes a theory of how problem solving is accomplished -- what mechanisms are common to all humans; what methods are possible and under what conditions they are evoked; and so on. This latter theory sometimes receives considerably less emphasis in discussions of cognitive simulation than the basic information processing model.

---

* Even though this term is currently used in a somewhat broader sense as a theory covering a miniature domain of behavior -- e.g., a theory of the T-maze.

The feelings of discomfort with program as theory are compounded by the difficulty of differentiating those parts of the program that have psychological import -- that are part of the theory -- from those that are only included to get the program to run on a digital computer. This is further compounded by the large size of simulation programs in numbers of instructions or subroutines, which seemingly imply a vast number of mechanisms, almost none of which have direct psychological support.

<u>The problem of program induction</u>. Observing current practice, one may ask where the simulating program comes from -- it appears to leap full grown from the head of some programming Zeus. While the question of how to induce programs from protocols has only minor relevance to validating theory, it is crucial to theory development. This is especially true, since we need to construct large numbers of microtheories in order to discover the general nature of the information processing performed by humans. That only a small number of simulations have actually been completed, each a product of excessive loving care, testifies to the need for further development of techniques for protocol analysis and program induction.

There seem to be several issues. Starting with the raw protocol, there is the question of how to extract information from linguistic utterances. The concern with linguistic data bequeathed us from the distrust of introspection by American behaviorism is subsiding and has been discussed elsewhere [4, 19]. But accepting the legitimacy of linguistic data does not of itself provide positive techniques for analysing them. Second, as already noted, simulation is often presented with only the basic theory of information processing described, the theory of problem solving going largely unmentioned. This creates the appearance that there are no guidelines about how to put a program together, only that one should start with a "symbol manipulating" system. Finally, there are few if any data-oriented techniques that permit the analyst to display the behavior of his subject so that the features that should be in the program become clear.

In this paper I will present one scheme for improving our ability to induce programs from protocols. By and large, the other issues will be ignored, although in the end some suggestions on assessment will emerge. The scheme will start from the data end -- from the protocol -- and gradually move toward completely specified programs, although never quite getting that far.

Theories of Problem Solving

We start with a brief restatement of the information processing theory of problem solving in a form that facilitates making contact with data from a new task. The theory, as sketched below, is not as broad in scope as the full range of experience in constructing programs to solve complex problems [20, 23]. However, it does appear to capture some of the central notions.

The theory assumes an underlying information processing system like that shown in Figure 4. This system comprises a large memory of symbolic structures, an essentially serial processor for accessing and restructuring this memory, and some imput-output structures. The organization is familiar enough, differing from existing hardware computers primarily in that (1) its memory organization is a constructable network of labeled associations between symbols, rather than a fixed numerically addressed array of words; and (2) primitive arithmetic processes are absent.

The detailed structure of the information processing system will be ignored. Providing that memory is sufficiently stable, the system is a universal machine, capable of carrying out arbitrary symbolic processes. Rather, the

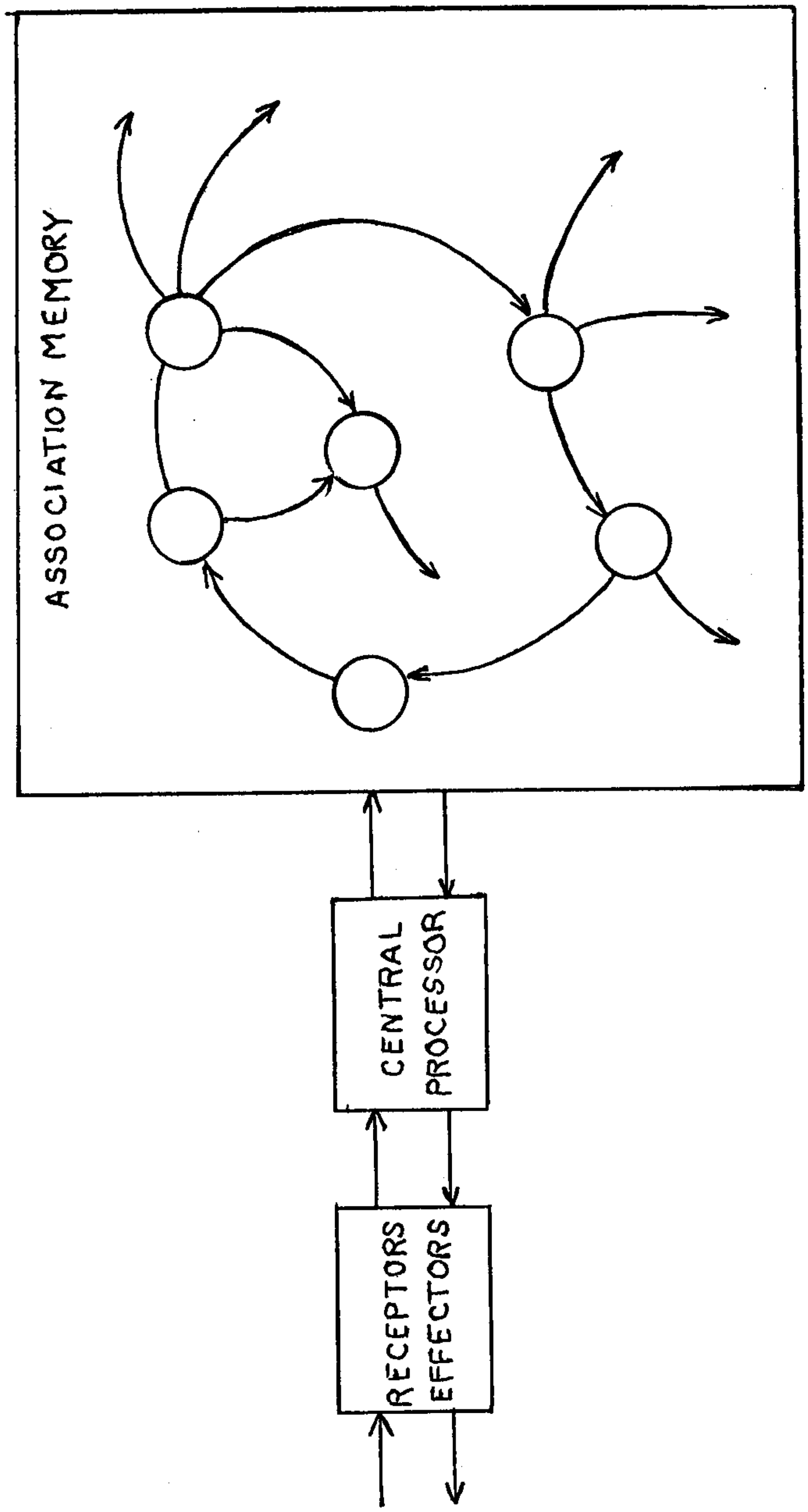


Figure 4: Basic information processing system.

theory is concerned with the methods, organization of processes, knowledge, etc., that constitute the program that a given system follows when problem solving. This viewpoint is clearly oversimplified. Limitations on immediate memory, on the rate at which reaction can occur to new data, and on the rate at which relatively permanent memory can be built up all pose boundary conditions within which the problem solving organization must operate. Additional limitations exist on perceptual and motor processes, but can be avoided in setting up experimental situations. Not so the former, and our excuse for ignoring them is our ignorance of how they affect problem solving, together with the fact that humans solve problems in such a way that these boundaries are not especially evident.

Problem solving takes place in a <u>problem space</u>. Abstractly considered, the elements of this space consist of <u>states of knowledge</u> about the problem. Operationally, they consist of data structures in the memory, which form an internal representation of the knowledge about the task environment. Both the <u>initial situation</u> and the <u>desired situation</u> must find their representation as elements of this space. Besides elements, a problem space has associated with it a set of <u>operators</u>, which, when applied to an element of the space, produce new elements. Thus, these operators are the means by which new information about the problem can be obtained from old.

Problem solving is always a matter of search -- of starting from some initial position (state of knowledge) and exploring until a position is attained that includes the solution -- the desired state of knowledge. The behavior of a problem solver is not fully determined by the problem space. Figure 5 shows the range of considerations that are relevant when the problem solver is at a position in the space. These express the degrees of freedom through

Evaluate new position:

Is it the desired state?

Should it be remembered, so that either can return to it later, or can recognize it when encountered again?

Is there some new information that should be extracted and remembered independently of position?

Is this progress, so that search should be continued; or are there difficulties?

Select new operator:

Has it been used before?

Is it desirable: will it lead to progress?

Is it feasible: will it work in the present position if applied?.

Apply operator to present position:

If works, then produces new position.

If not work, what are the difficulties?

Evaluate difficulty:

Should a subgoal be set up to overcome this difficulty?

Should the position be rejected?

Return to prior position?

Return to initial position?

Return to a remembered position; if so, which one?

Evaluate old position, just returned to:

Should it be used, or rejected?

Figure 5: Considerations at a position in problem space.

which intelligence (or stupidity) can be manifested. The considerations of Figure 5 do not form a program for behavior at a position, since the system of a problem solver may organize them very differently, perhaps ignoring some altogether. Nor is the list necessarily complete, although it seems to encompass many of the considerations used by both artificial and human problem solvers.

Search is a problem space is <u>constructive</u>. The elements of the space, although they exist abstractly, do not exist for the problem solver unless he generates them, or remembers them for later retrieval once generated. This gives the search a different character from that through a world that exists independently of the problem solver -- e.g., a forest. In essence, problem spaces are always exponentially growing trees: two independent paths cannot end up at the same element of the space. One cannot do in a problem space what one does in a forest: put marks on trees to recognize the same place if it is returned to. In the problem space a data structure may be generated that is identical in structure and content to another -- but it will not be the same data structure, hence will not contain any "tree mark." Only if the problem solver remembers each new element as it is constructed, and determines if each new one is identical with any of those kept so far, will he be able to simulate the tree marking scheme.

Initially, a problem solver is given a problem through some external representation of the pertinent situations, goals, constraints, conditions, operations, auxiliary facts, etc. The problem space is <u>not</u> given -- the problem solver must select or create a problem space in which to solve the problem. That is, he must encode the information in the external representation into an internal one in which he can effect the transformations required by the operators, which he also constructs (or selects). This problem space may be already available inside the problem solver -- he may simply translate into an already well known system. Alternatively, it may be constructed out of more elementary things he can do, as when he learns a new set of operations provided by the experimenter.

Currently, the theory says little about the selection and construction of problem spaces; primarily because experience so far has been mostly with problem solving systems in which the investigators invented the problem spaces themselves and simply programmed the computers to problem solve in them. As we shall see, the question of what problem space is used is critical. However, it should not be assumed that the problem spaces used are exotic. They often lie very close to the obvious one suggested by the defining conditions of the problem.

The problem solver is not limited to a single problem space. He may obtain a new one after finding the initial one inadequate. More important, he may make use of more than one simultaneously. An example is provided by the program for proving theorems in plane geometry [11], which uses both a space of symbolic expressions, representing theorems, and a space of coordinates, representing the diagram. This latter provides much of the problem solving power of the system, since operations of direct measurement of angles and length are available in it to check the assertions of the theorems.

The possibility of using several problem spaces emphasizes that the total problem solving system is not to be simply identified with a single problem space. Information that is constant throughout a problem may find no representation in the state of knowledge, nor will the processes that take it into account. Retrieval processes and the organization of large amounts of data may not be represented in a problem space, even though of critical importance to problem solving.

The Problem Behavior Graph (PBG)

Let us see what this theory implies when applied to protocol material. If we knew what problem space the subject was working in, then we could view his behavior, as revealed through the protocol, as a search in this space. More precisely, we would be able to 1) state the kinds of information that make up the states of knowledge of this space; and 2) specify a set of operators, such that each change in the state of knowledge corresponds to an application of one of the operators.

From a descriptive point of view we can ignore all of the considerations of Figure 5. To track the subject's search it is enough to have well specified just the elements and operators of the problem space, not all the additional rules of selection and decision. Even so, we have stipulated a non-trivial requirement. Numerous cues exist in any protocol about both the state of knowledge and the operations and inferences the subject is performing -- the language is full both of phrases indicating propositions and phrases indicating processes and actions. Since the set of operators is fixed, and since every change in state of knowledge is to come about through the application of one of these operators, there are many places to go wrong.

The actual problem space used by the subject is unknown. Indeed, it is even unknown if the subject is behaving in accordance with the theory. Consequently, the appropriate data analysis procedure is to posit a problem space and see if the subject can be analysed as searching in this space.

In case the subject is wandering in more than one space, of course, the two must be unravelled simultaneously. If we are successful, we shall know it by getting a reasonably complete picture of the search (it will not be perfect in any event due to ambiguity and incompleteness in the protocol). Then, we can go on to consider what other information about the remainder of the subject's program can be obtained.

Search trees published in the literature of problem solving programs show mostly the total extent of the search -- what positions were ultimately visited [12, 28]. Often, if the search strategy is simple -- e.g., a so-called depth-first strategy -- the actual path of search can be inferred from the total tree. However, we need a way of tracking the search that lets us reconstruct the time history. The scheme we adopt we call the Problem Behavior Graph (PBG). We give the conventions below; referring to Figure 6 for an example.

Rules for Problem Behavior Graph (PBG)

A state of knowledge is represented by a node (the labeled boxes in the figure).

The application of an operator to a state of knowledge is represented by a horizontal arrow to the right; the result is the node at the head of the arrow (Operator Q1 to position P1 gives position P2).

A return to the same state of knowledge as node X is represented by another node below X and connected to it by a vertical line (P3 results after abandonment of P2; it constitutes the same state of knowledge as P1).

Time runs to the right and down; thus the graph is linearly ordered by time of generation (from P1 to P5).

The problem solver is viewed as always being located at some node in the PBG, and having available exactly the information contained in its state of knowledge. The act of search itself generates information in addition

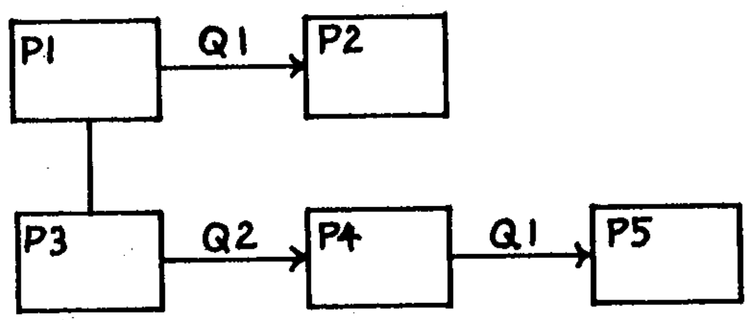


Figure 6: Problem Behavior Graph (PBG).

to that represented at the node: in particular, path information about how the node was arrived at; and past attempts information about what else has been done when in this state of knowledge. Both these kinds of information are viewed as being associated with a node; in fact, this sort of information is what distinguishes node P3 from P1.

With this much apparatus, we are ready to consider some examples.

Crypt-arithmetic. The top of Figure 7 shows a version of a familiar puzzle, called a crypt-arithmetic problem by one collector [2]. Each letter is to be assigned a distinct digit between 0 and 9 such that when the letters are replaced by their assigned digits a legitimate sum is obtained.
As a starter, it is given that D is 5; thus, no other letter can be 5 and a 5 must replace all three occurrences of D in the figure.

In accordance with the paradigm, a subject (a college student) was given the task to solve, with instructions to "think aloud". The initial segment of his protocol is shown in Figure 8. It has been broken into short phrases, which have been labeled. The segment shown amounts to about 12% of the total protocol, the last phrase of the full protocol being B321 (the subject solved the problem). The expressions on the right side of Figure 8 will be discussed later.

The first step in the analysis after obtaining the protocol is to construct a problem space. The simplest one, of course, is defined directly from the rules of the puzzle. The elements are sets of assignments; the operators are the acts of assigning a new digit to a new letter. The initial position is that one where no assignments have been made; and the final position is the one where all ten have been made, such that the three constraints have been satisfied. In fact, this problem space would be used by someone who wanted to build a simple search program for the task. Clearly, our subject is more

Problem: DONALD +GERALD ROBERT   D←5   Each letter assigned to one and only one digit. Each digit assigned to one and only one letter

```
  DONALD
 +GERALD
  ROBERT
```

Terms: entities that can be referred to in problem space

- l is any letter, A, B, D, E, G, L, N, O, R, T
- d is any digit, 0, 1, ..., 9
- ds is any set of digits, d,d,...,d
- c is any column, c1, c2,..., c7 (c1 is the right hand column)
- t is any carry to a column, t1, t2,..., t7
- v is any variable, either a letter, l, or a carry, t

Elementary expressions: relationships and properties amond terms

v←d	v has been assigned the value d
v=d	v has the value d by inference
v=ds	v has one of the values in the set ds
l>d	l has the respective constraint
l<d	
l even	
l odd	
l free	l can take any value (in an implied domain) without constraint

Expressions: an elementary expression or term, ee, followed by a suffix

ee-p	ee is not possible or can take no possible value
ee?	the truth or value of ee is unknown
ee!	the truth or value of ee is critical to the inference

States of knowledge: any conjunction of expressions (need not be consistent)

Operators

PC(c) Process the column c. The input is all the information about the column and the letters and carries in it; the output is some information that can be inferred from the column, which may include specification of something as critical (!) or unknown (?).

GN(v) Generate the values of variable v. This takes into account the constraints known to hold for v (e.g., v odd), but not the exclusion of values due to assignment to other variables.

AV(v) Assign a value to the variable v. The output is in form v←d. This value will be selected from the set generated by GN(v).

TD(l,d) Test if l can take the value d. Failure is due to d being assigned to another letter, or to d lying outside the permissible range for l.

Goals

get v	get a value for v; determine something about the value of v
get ee	determine whether expression is true
check ee	determine whether expression, known to be true, is in fact true

Figure 7: Crypt-arithmetic: Definition of problem space.

B1	Each letter has one and only one numerical value --	? :		(ask Exp. about rules)
B2	Exp: One numerical value.			
B3	There are ten different letters			
B4	and each of them has one numerical value.			
B5	Therefore, I can, looking at the two D's --	S1:	$D \leftarrow 5$	$\rightarrow$ FC(D)=> c1; PC(c1)=> T=0
B6	each D is 5;			
B7	therefore, T is zero.	T1:	T=0	$\rightarrow$ TD(T,0)=> +
B8	So I think I'll start by writing that problem here.			
B9	I'll write 5, 5 is zero.			
B10	Now, do I have any other T's?	S1:	T=0	$\rightarrow$ FC(T)=> $\emptyset$
B11	No.			
B12	But I have another D.	S1:	$D \leftarrow 5$	$\rightarrow$ FC(D)=> c6 (no PC(c6))
B13	That means I have a 5 over the other side.			
B14	Now I have 2 A's	G4:	get 1s	$\rightarrow$ FL(1s)=> R; get R
B15	and 2 L's			
B16	that are each --			
B17	somewhere --			
B18	and this R --			
B19	3 R's --			
B20	2 L's equal an R --	S2:	get R	$\rightarrow$ FC(R)=> c2; PC(c2,R)=> R odd
B21	Of course I'm carrying a 1.			
B22	Which will mean that R has to be an odd number.			

Figure 8: Crypt-arithmetic: Initial segment of protocol.

B22.1		R1:	PC unclear → get R; repeat PC	
B23	Because the 2 L's --	↑ :		PC(c2,R)=> R odd
B24	any two numbers added together has to be an even number			
B25	and 1 will be an odd number.			
B26	So R can be 1,	S4:	get R	→ GN(R)=> 1,3,5,7,9
B27	3,			
B28	not 5,	T1:	R=d	→ TD(R,d)=> R=5-p(D←5!)
B29	or 7.			
B30	or 9.			
B30.1		? :		
B31	Exp: What are you thinking now?			
B32	Now G --	S2:	get R	→ FC(R)=> c6: PC(c6,R)=> G even
B33	Since R is going to be an odd number			
B34	and D is 5,			
B35	G has to be an even number.			
B35.1		R1:	PC unclear → get G; repeat PC	
B36	I'm looking at the left side of this problem here where it says D + G.	↑ :		PC(c6,G)=> t6?
B37	Oh, plus possibly another number,			
B38	if I have to carry 1 from the E + O.			
B39	I think I'll forget about that for a minute.	? :		
B40	Possibly the best way to get to this problem is to try different possible solutions.			
B41	I'm not sure whether that would be the easiest way or not.			

Figure 8 (continued)

sophisticated. He makes inferences using the column constraints; he uses the carry; he works with concepts such as even-oddness; he attends to the columns in variable order.

The bottom part of Figure 7 provides a definition of a problem space for this subject*. The element, corresponding to the state of knowledge, is a conjunction of elementary expressions, each of which deals with some relation between variables (letters or carries) and digits. Neither path information nor past attempts information is stated explicitly. Actually, we would hope to infer from the PBG what information of this kind is being kept.

There are four operators**. Each is defined with reasonable precision in terms of input-output characteristics, which are the features necessary to identify whether the operator was evoked in the protocol. Whether all occurrences so identified constitute a single operator, in the sense of being produced by a consistent subroutine, is a matter for later analysis***. The initial part of the PBG, extending somewhat beyond the segment of protocol reproduced in Figure 8, is given in Figure 9. The double lines indicate that an operator is being repeated from the same state of knowledge. A condensed version of the complete PBG is given in Figure 10.

* This analysis of crypt-arithmetic is taken from [22], where the entire protocol and all the other matters dealt with here informally are treated in greater detail. In particular, the problem space is defined by means of Backus Normal Form, in order to give a precise description of what information can constitute a state of knowledge. Clarity, of course, is essential if the concept of state of knowledge is to be more than a descriptive metaphor.

** To the alert reader: The formulas on the right hand side of Figure 8 contain not only the four operators of the problem space, but others as well, which will be discussed later.

*** It is not discussed further here, but see [22].

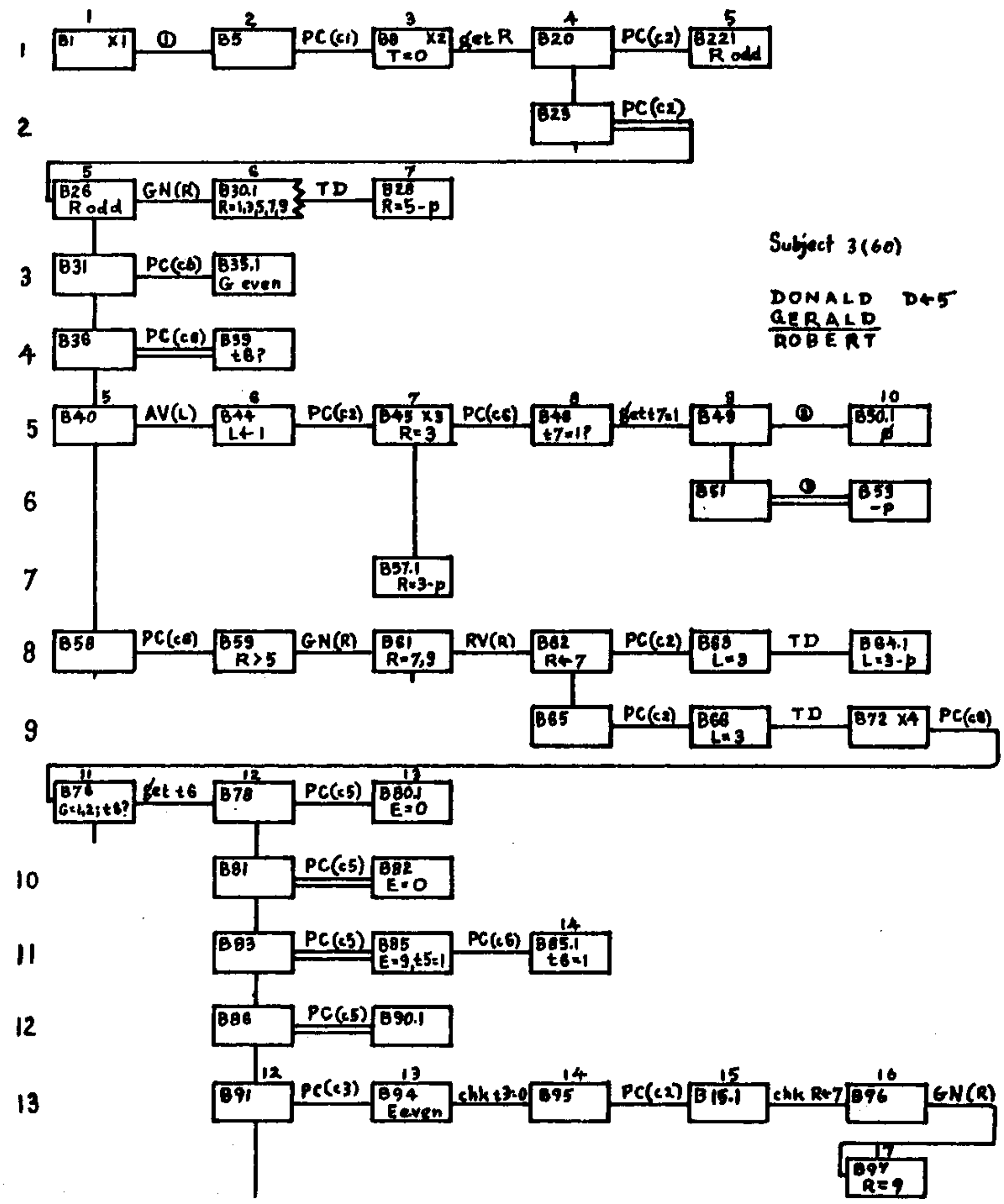


Figure 9: Crypt-arithmetic: Initial segment of Problem Behavior Graph.

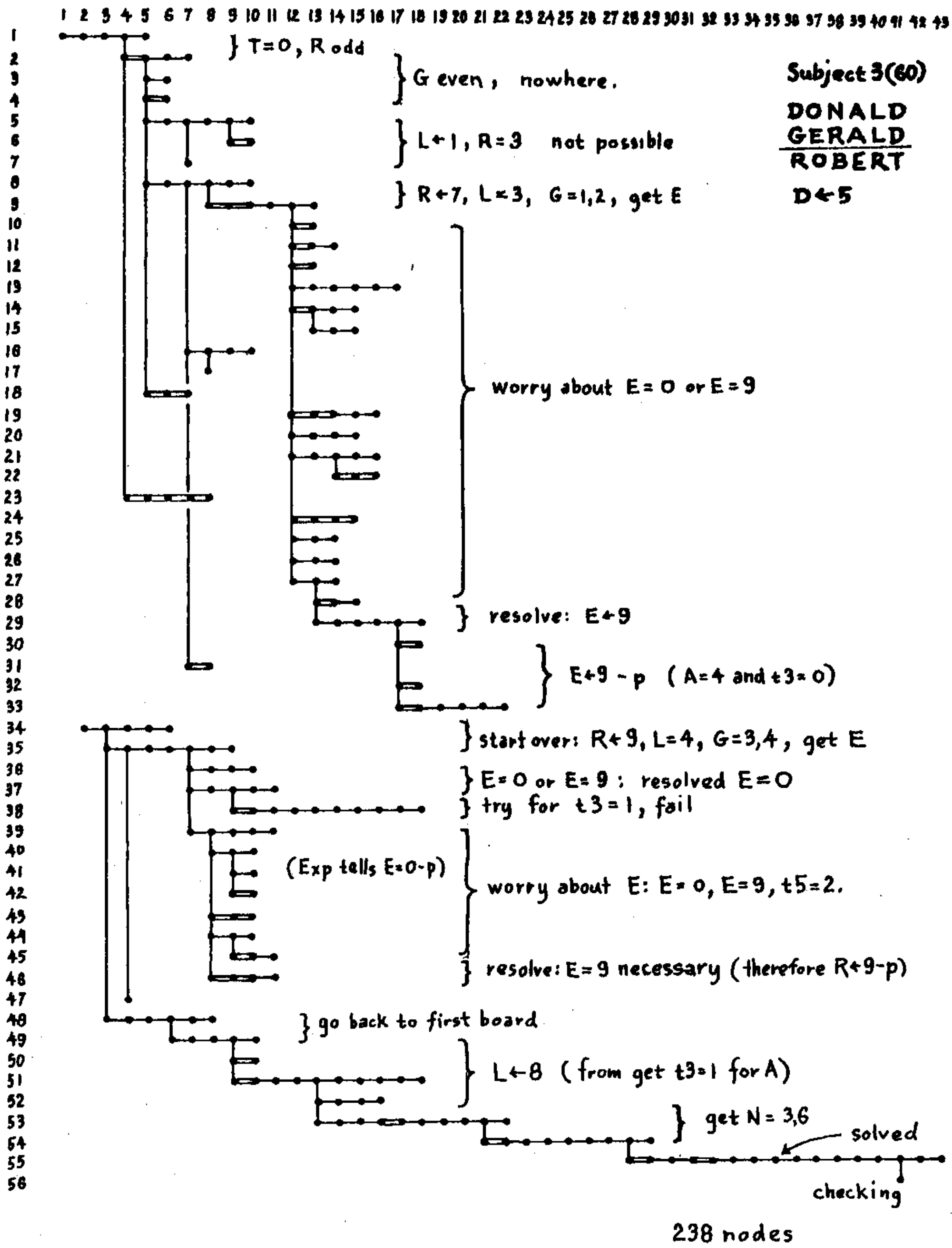


Figure 10: Crypt-arithmetic: Total Problem Behavior Graph.

Let us consider briefly how the coding goes. Starting at the beginning (B1) we have an exchange that is really outside the problem space, since it involves clarification of the rules. We simply indicate this by a special footnote (1). In the second box, B5, we have a clear statement of 1) considering the two D's, asserting their value, and concluding that T is zero. The coding of this as the operator PC(c1) is clear. Some open questions are 1) when did the inference actually occur; 2) why did column c1 get considered; 3) was it desired to find the value of T before processing column c1; and 4) was it also concluded that t2=1 ? About some of these questions we do not need to have the answers. As to the first, we require only the approximate ordering. As to the second, the selections of columns is internal to each box and thus irrelevant to the problem graph. The third question is relevant, but we adopt the view that unless specific information is available on the variable desired, we will not record it. Finally, although it is plausible that t2=1 is inferred since 5 + 5 = 10, there is no immediate evidence. However, later behavior (B21) shows that in fact this information was retained.

The next box, B8, should be considered in conjunction with box B20. In this latter we clearly have a consideration of column 2 with the inference of R odd. If we write down what happens before this we have:

B8-B9	Writing prior result
B10-B11	Searching for a next step with no result in terms of our problem space.
B12-B13	Another writing step, when D of c6 is noticed; conceivable that new information obtained, but certainly no evidence for it. (Result of B9 and B13 indicated by X2.)
B14-B19	Consideration of c2, c3, A, L and R in the apparent search for a next step. No new information obtained in our problem space.
B20-B22	Processing of c2.

The concern with R, clearly indicated in B18 and B19, leads to the inference that the decision to process column c2 is based partly on the decision to obtain some information about R. Thus we code B8 with the goal of getting R. Those things occurring prior to B18 all belong within a box: the operations of writing and the (attempted) selection of columns on which to work. If the inference to get R were less clear, we would have only a single box for B8 to B22, whose operator would be PC(c2).

It is clear that in B23 to B25 the reasoning used in B20 to B22 is repeated. Why the repetition occurred is not as clear. It might be to check the processing -- to assure that the inference is correct. That a correction can occur the second time around is shown by the sequence B32-B35, yielding G even, and the immediate repeat, B36-B38, leading to the realization that no such inference is possible. Repetition might also be affected by the experimental instruction to get the subject to talk. In any event, we need to create a box, B22.1, for the result of the first PC(c2) and then back up one for the second at B23.

In B26-B30 an explicit generation of the odd digits follows immediately upon the (confirmed) conclusion that R is odd. Thus the inference that GN(R) occurred is not problematic. The generation does not take into account what values are already used, since the already used digit, 5, is generated and explicitly rejected. This supports the inference that TD was applied to the output of GN. It is not as clear, of course, that TD was applied to 1, 3, 7 and 9, since these were OK and no special indication of their acceptability is provided. However, if TD was applied sometimes and sometimes not, then a process must have existed to make this decision; but this process would have had to perform (uniformly) the same function as TD; namely, to determine if a digit were used. Consequently, it is simpler to assume that TD was applied uniformly.

B31 signals a pause, since the experimenter breaks in with a prod to talk. Since there is no evidence in what follows B32 that the refinement of the information to R=1,3,7,9 is used, rather than the more primitive, R odd, it is inferred that the search backed up. Quite possibly additional processing did go on from B30.1 during the pause, but since we have no evidence for it, we make no explicit note of it. If new information were obtained, it should show up either at B31 (which it doesn't) or at some later time as new "unexplainable" knowledge.

We have only given the first bit of a very long (and dull) argument. In a majority of cases the encoding is quite clean. Frequently, some appreciable inference must be made as to the underlying process. And in a few cases we have no information as to what transpired, as at B30.1. The basis of these inferences, from the most obvious to the most indirect, lies in our (the encoder's) ability to interpret natural language. This interpretation itself demands, however, a view of the task in information processing terms and of the subject as an information processing problem solver. Thus, we have not attempted any encoding of the language of the protocol prior to extracting the PBG. Where such an <u>a priori</u> coding is possible, e.g., "each D is 5; therefore, T is zero", it isn't needed. Where it is needed, "Now I have 2 A's and 2 L's that are each -- somewhere -- and this R -- 3 R's -- ", it isn't safe*.

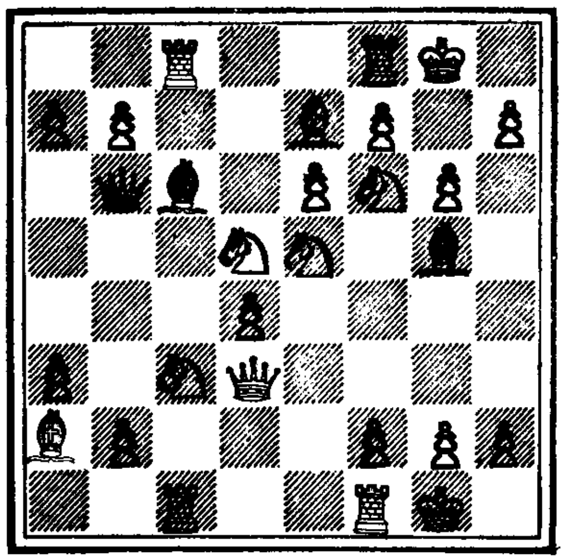
---

* In an earlier study of chess [27] we did try a preliminary coding, but achieved little more benefit from it than the segmentation of the protocol into elementary phrases.

What do we learn from the PBG for this subject? First, his problem solving can be described as search in a well defined problem space. Second, from the definition of the problem space, we obtain information about the intellectual tools he is capable of using. This is revealed most clearly by the kinds of situations in which PC is able to provide new information -- e.g., to take as inputs R odd and D←5 and produce G even. Third, we have taken a preliminary step to asking if there exist regularities in his search behavior. This does not follow from the existence of the search tree. The encoding has been done entirely on a local basis. Whether the subject has consistent modes of behavior for carrying out the considerations of Figure 5 remains an open question. The PBG does provide a segmentation of the total stream of behavior into a set of units (238 of them in this case) that now permit inquiry into further regularities. Before turning to this next stage of analysis, let us examine some PBG's in other tasks.

<u>Chess</u>. Figure 11 shows a complex middle game position in chess, and Figure 12 gives the initial segment (15%) of the protocol of a player of moderate ability choosing a move for White. The problem space for this subject, shown in Figure 11, is by and large the obvious one. The elements are chess positions, the operators are moves. The position of Figure 11 is the initial position and the subject searches out from it looking for positions of advantage. These latter are characterized mostly by tests and not by specific chess positions. But these tests are only brought to bear on positions constructed via moves from the initial position, so that a more generalized problem space element is not required.

The subject occasionally makes use of generalized operators of the form "Man M on Square S defends," or "Man M on Square S moves away." That is, the man is fully specified, but the square to which he moves is only specified up to a function term. This does imply a generalization of the concept of position,



Position A [4]

White to move.

<u>State of knowledge</u>. A generalized chess position containing:

<u>Located men</u>: Men are located on a specific square

<u>Unlocated men</u>: Men on no specific square, but have an associated function they have performed

<u>Functions</u>: Functions that have been performed, but whose agents have not been specified

<u>Operators</u>: A move consisting of an agent (the man being moved) and an action (his moving to a square, vacating a square and perhaps capturing a man)

<u>Legal moves</u>: The agent is a located man and the action is specified

Examples: Given in standard chess notation, except that on the graphs Black men are primed

P-K4 White Pawn moves to square K4

Q'-Q1 Black Queen moves to square Queen 1

<u>Function moves</u>: A located man as agent with the action given only by the function to be performed; produces an unlocated man

Examples: B-defends

Q'-retreats

An action either specified completely or by function, but with no agent given

Examples: wxB White capture a specific Black Bishop

not P'xP A Black move that is not a specific one

Figure 11: Chess: Definition of problem space.

E1 B1 Okay, White to move...

B2 In material the positions are even.

B3 One, two, three, four, five six - six Pawns each.

B4 Black has what threats?

B5 His Queen is threatening my Knight's Pawn

B6 and also he has one piece on my Queen's Pawn -

B7 has a Rook in front of the Bishop,

B8 which will give him an open file.

B9 Let's see, all right, what threats do we have?

B10 We have his Knight under single attack

B11 protected by the Bishop.

B12 We have his other Knight under attack

B13 protected by three pieces.

B14 The Queen is bearing down on the Knight's Pawn

B15 and the Rook is over here protecting the Knight

B16 and the Bishop at Rook 2 is bearing down on the Knight.

B17 All right, looks like we have something going on the King's side.

B18 All Black's pieces are over on the Queen's side -

B19 most of them out of play -

B20 good chances for an attack perhaps.

E2 B21 See, what moves are there?

B22 The Bishop at Rook 2 can take the Knight,

B23 which would be no doubt answered by either Bishop takes Bishop or Pawn takes Bishop.

B24 Probably Bishop takes Bishop

B25 to avoid isolating the Pawn.

B26 If we then play Knight takes Bishop.

B27 he will then play Pawn takes Knight or Rook takes Rook,

B28 but this would give White an open file if he exchanged

B29 and this is doubtful.

B30 This would isolate Black's Queen's Pawn -

B31 it would be protected only by the Knight

B32 which is pinned,

B33 therefore we could move the Queen to Bishop 3,

B34 not only putting another threat on the Knight,

B35 but also threatening an isolated Pawn.

B36 Both of them could not be protected simultaneously unless Queen to Queen 1.

E3 B37 All right, well, what about Queen to Bishop 3 immediately.

B38 Queen to Bishop 3 immediately is not good -

B39 it gives no threat on the Knight at Bishop 3

Figure 12: Chess: Problem segment of protocol of subject 2.

since after such a move has been made the board is not fully specified. That the subject can take such partially specified boards and apply other chess moves (operators) to it shows that the problem space is genuinely larger than the space of chess positions. This corresponds in crypt-arithmetic to the expansion of the state of knowledge to include subset information -- e.g., that R is odd or that E is 0 or 9. Although the problem space includes function moves on located men, it does not include moves on unlocated men; e.g., moving a man who was previously "moved away."

The PBG for the subject's total analysis, which lasted about 17 minutes, is shown in Figure 13. This is taken from a previously published paper [27]. It can again be inferred that the subject's behavior can be viewed as search in a well defined problem space. Further, it is clear that the subject does not reason very abstractly about the position; his tools of analysis focus on the exploration of specific future paths. These tools include the range of functions indicated in Figure 5 -- move generators, evaluation functions, etc. -- but they still work within this highly concrete framework.

As in the case of the crypt-arithmetic example, the chess PBG provides a segmentation of the total behavior in a form in which further regularities can be sought. However, even without detailed examination, the total graph reveals a striking regularity: the search proceeds by a series of deep penetrations with very little branching (and only first level branching), followed by a return to the initial position (all of the base points on E1 to E25 the initial position although not tied together by a vertical line). Following DeGroot [4], we have called this the <u>progressive deepening</u> strategy of search. More details can be found in the original paper.

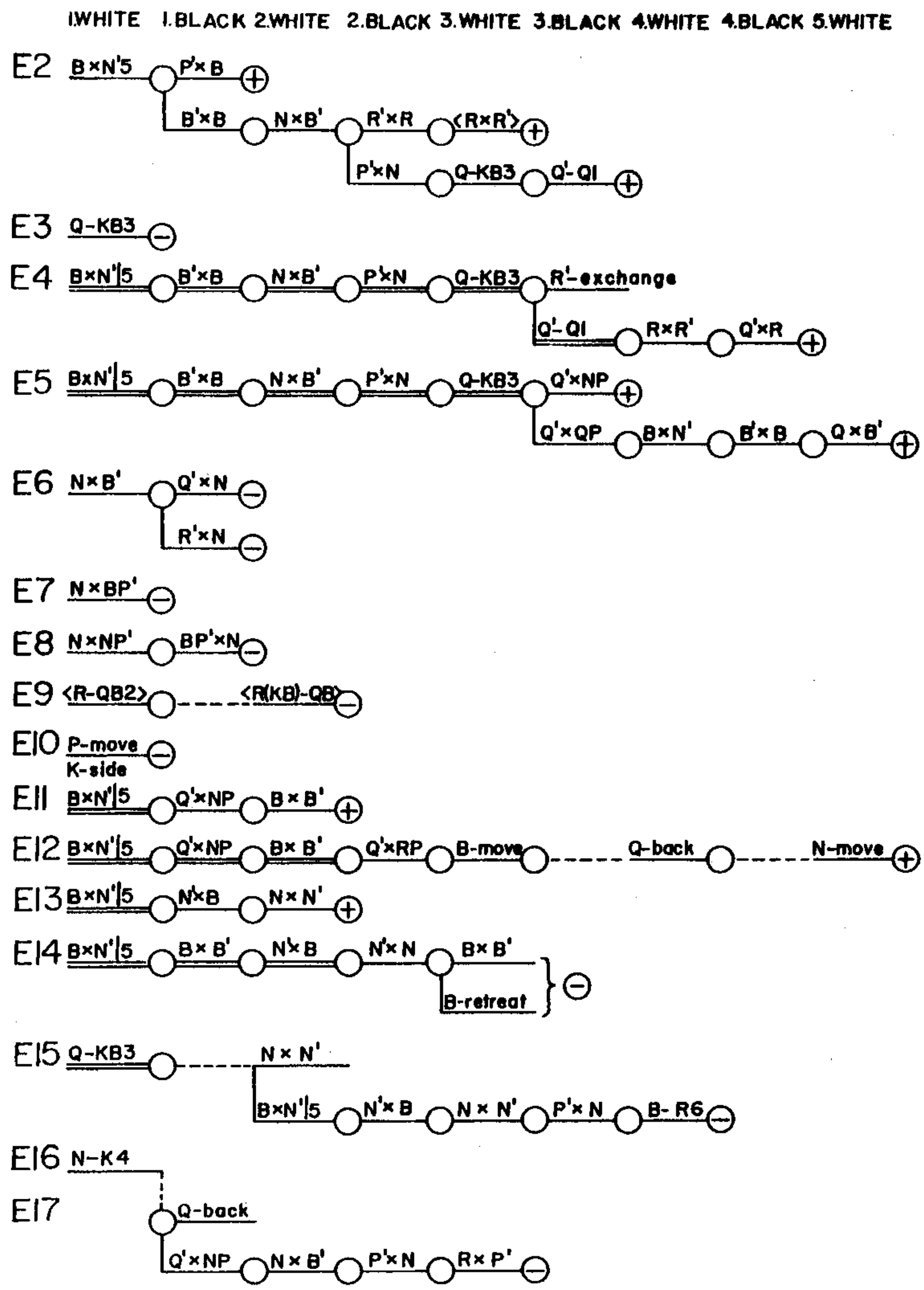


Figure 13: Problem Behavior Graph of Subject 2 [reproduced from 27].

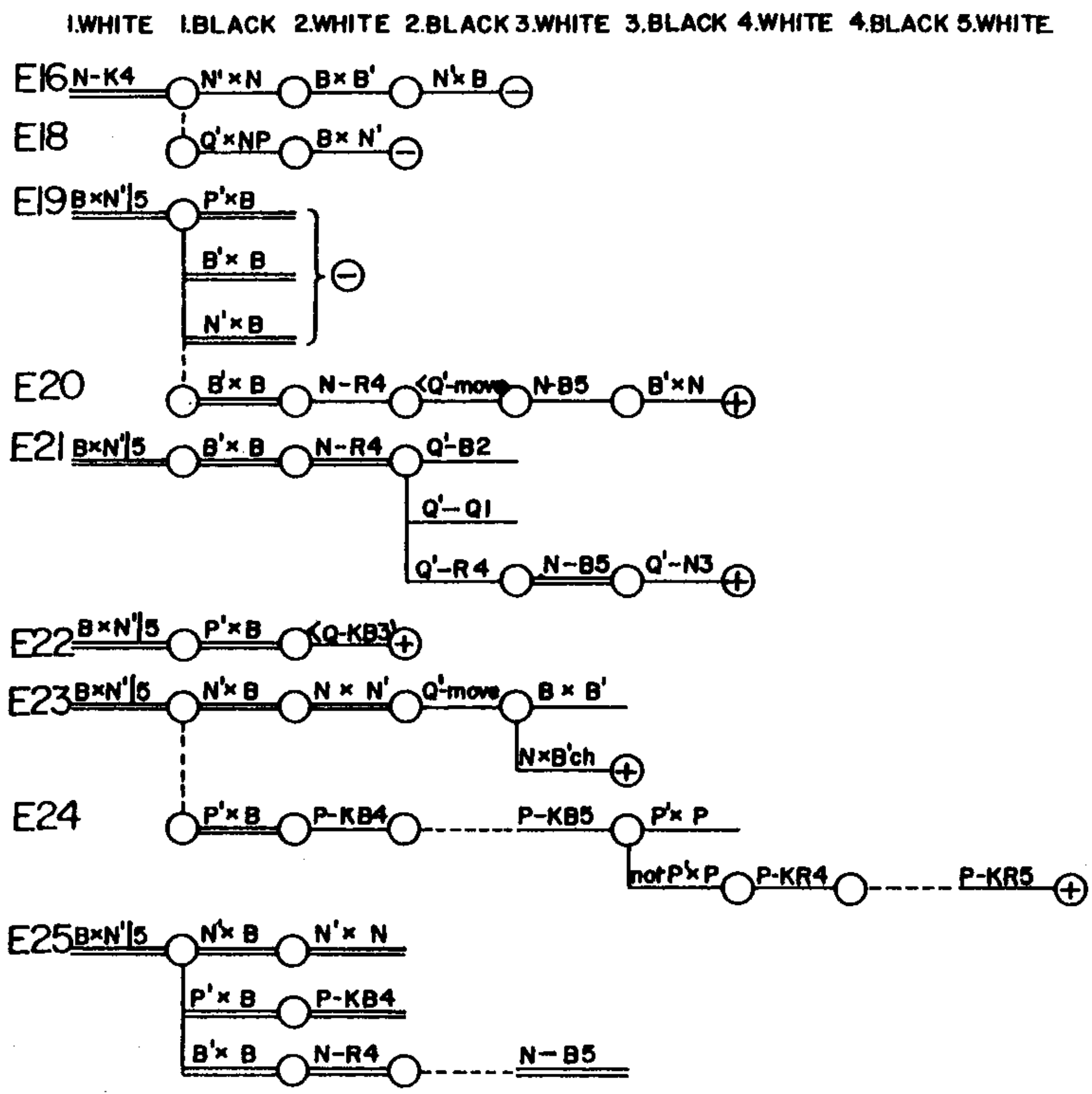


Figure 13 (continued)

No such clear cut strategy shows up in crypt-arithmetic example. Partly this is because of the forms of external memory available. In the chess of Figure 13 only the initial position is available; all other positions must be carried in the head. In the crypt-arithmetic of Figure 10 the subject is permitted write and erase operations in an external memory. Thus, he could go back not only to the initial situation, but also to the one written on the board.

The position of Figure 12 is taken from DeGroot [4]. Hence, it is possible to go back and reanalyse some of his protocols on the same position. Figure 14 shows the PBG of Max Euwe (world chess champion, 1935-37). It should be noted, however, that Euwe was by far the most methodical of the grand masters studied by DeGroot, and the one who produced the most copious protocols.

Logic. Figure 15 shows the PBG for the logic problem presented in the original example. The behavior in Figure 3, corresponds to the first line (where it is assumed that both parts of R8 are carried along together); the simulation reported in [24] was carried through line 5. The full PBG represents the total episode, lasting close to thirty minutes and ending in the subject finding a solution.

The basic problem space is that defined by the experimenter in setting up the task. The states of knowledge are the sets of expressions that have been derived to a given point. The operators are the 12 rules in Figure 1 (actually representing a very large number of operators if all variations are taken into account). The initial element of the problem space consists of the single expression initially given (in other variants, several initial expressions were used); the desired situation is given explicitly.

The subject modifies this basic space in two ways. First, in the same manner as in the crypt-arithmetic example, he works between two spaces --

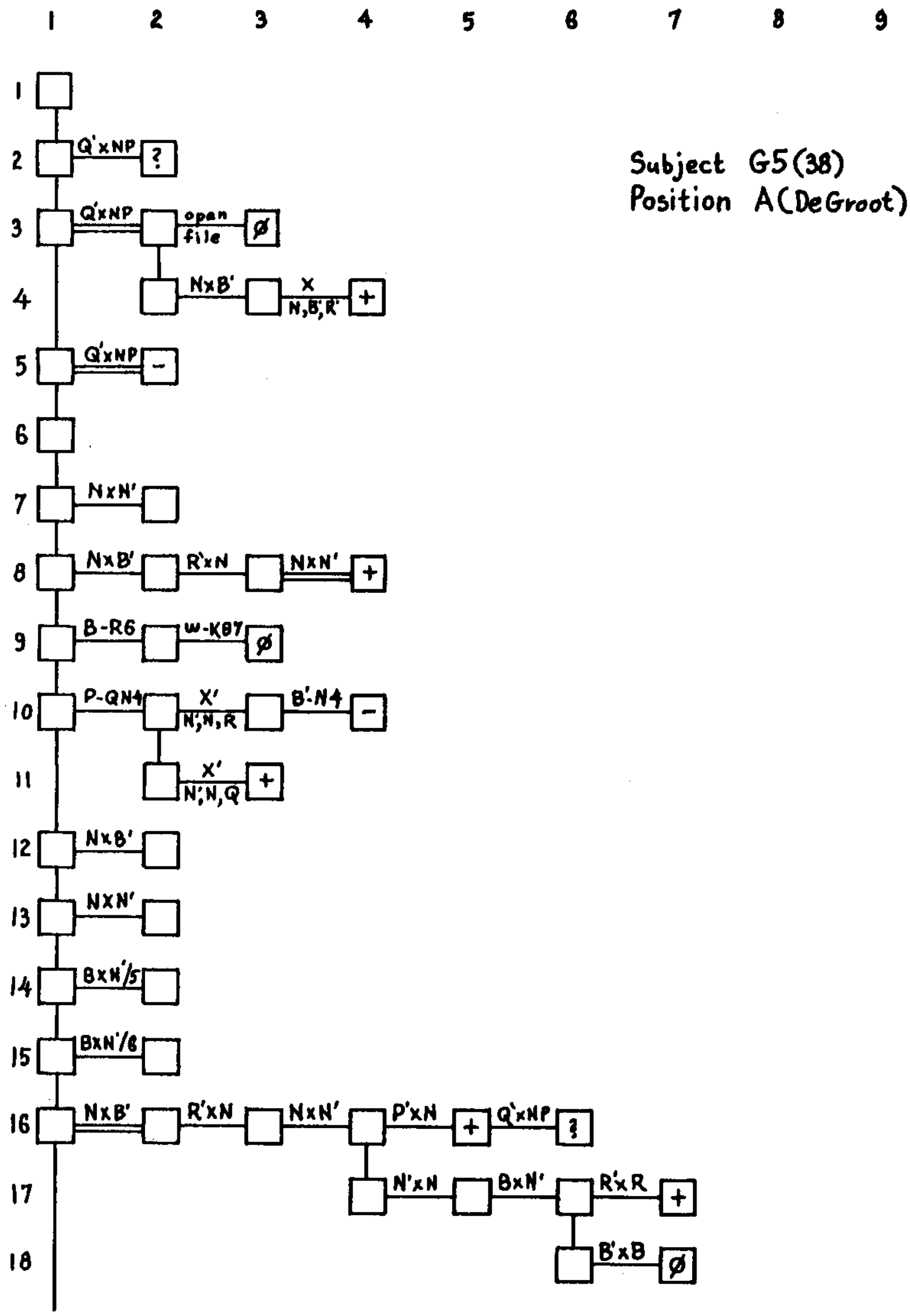


Figure 14: Chess: Problem Behavior Graph of M. Euwe [from protocol in 4].

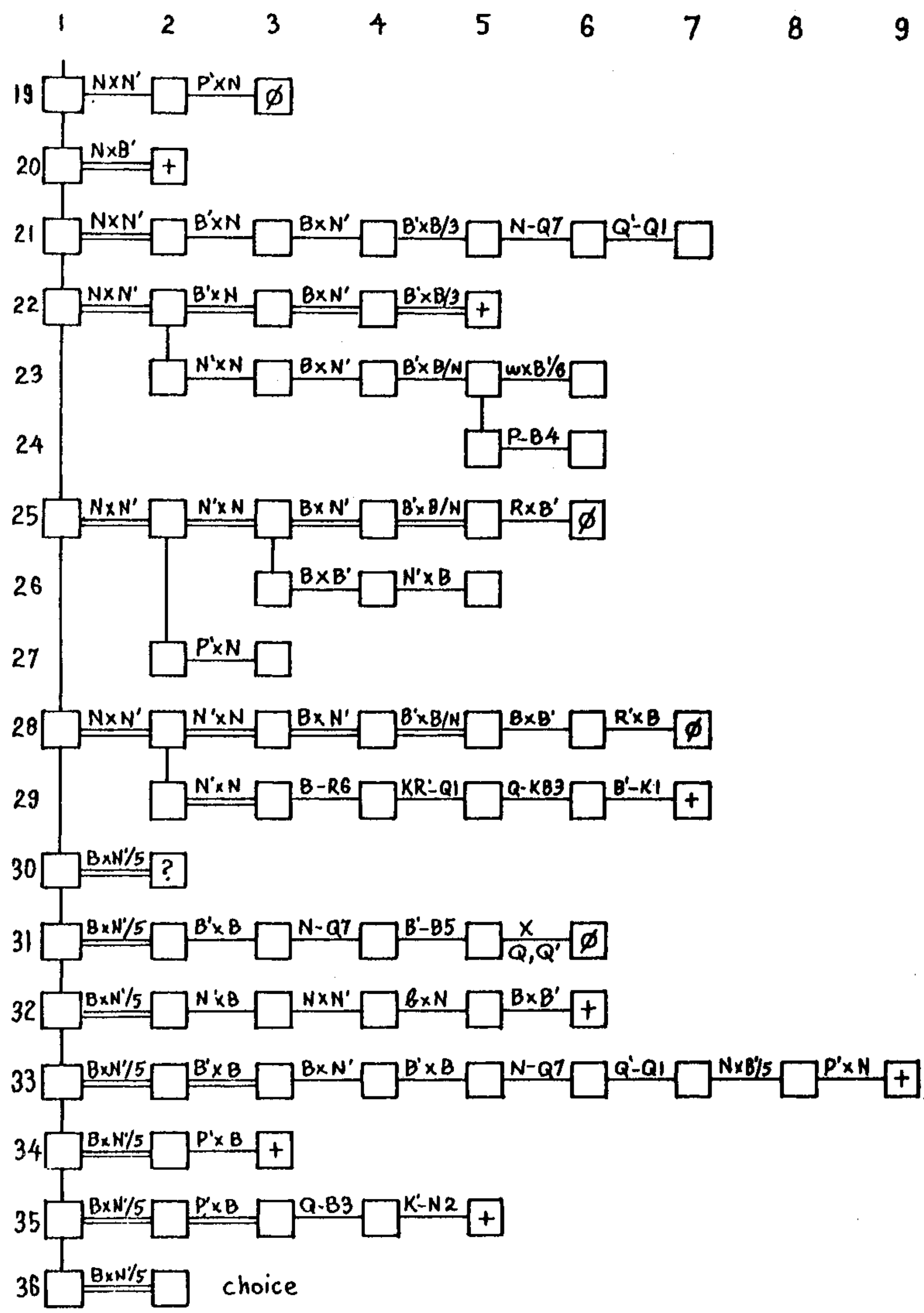


Figure 14 (continued)

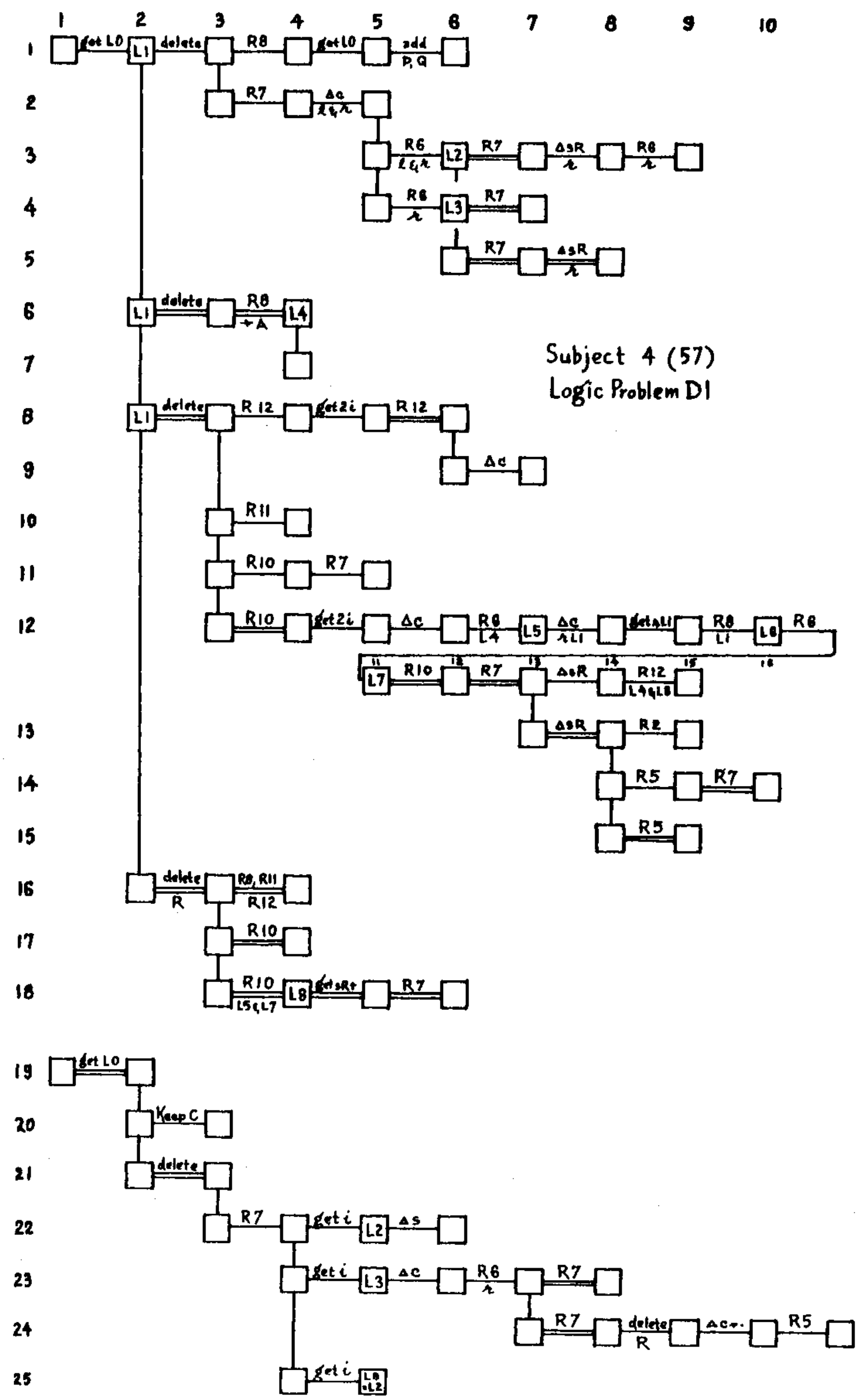


Figure 15: Logic: Problem Behavior Graph of subject 4 on problem D1.

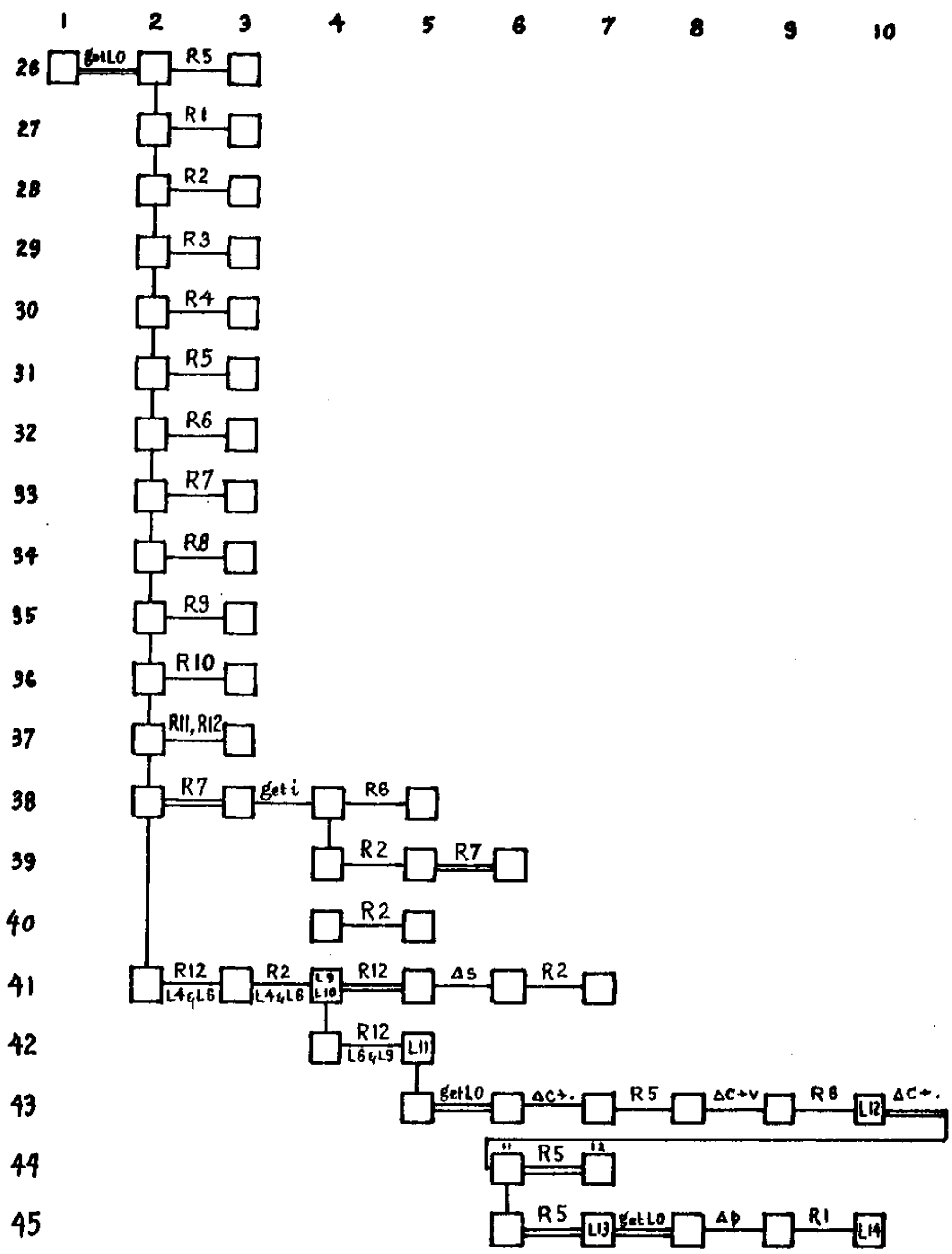


Figure 15 (continued)

the written one and the one in his head. Thus, although by the rules of logic anything derived becomes part of the current state of knowledge forever, the subject cannot remain cognizant of the entire past. Instead, the current state is defined by a subset of those expressions that have been derived. This is forced on the subject by working internally. However, it may even be true of some of the expressions that have been recorded on the board; they may be taken as irrelevant and not enter into the processing of the current state. The extent and lawfulness of these constrictions of the actual state of knowledge from that available according to the permissible rules of inference is a matter for later analysis. What is recorded in Figure 15 is the information actually used in advancing the search at each point.

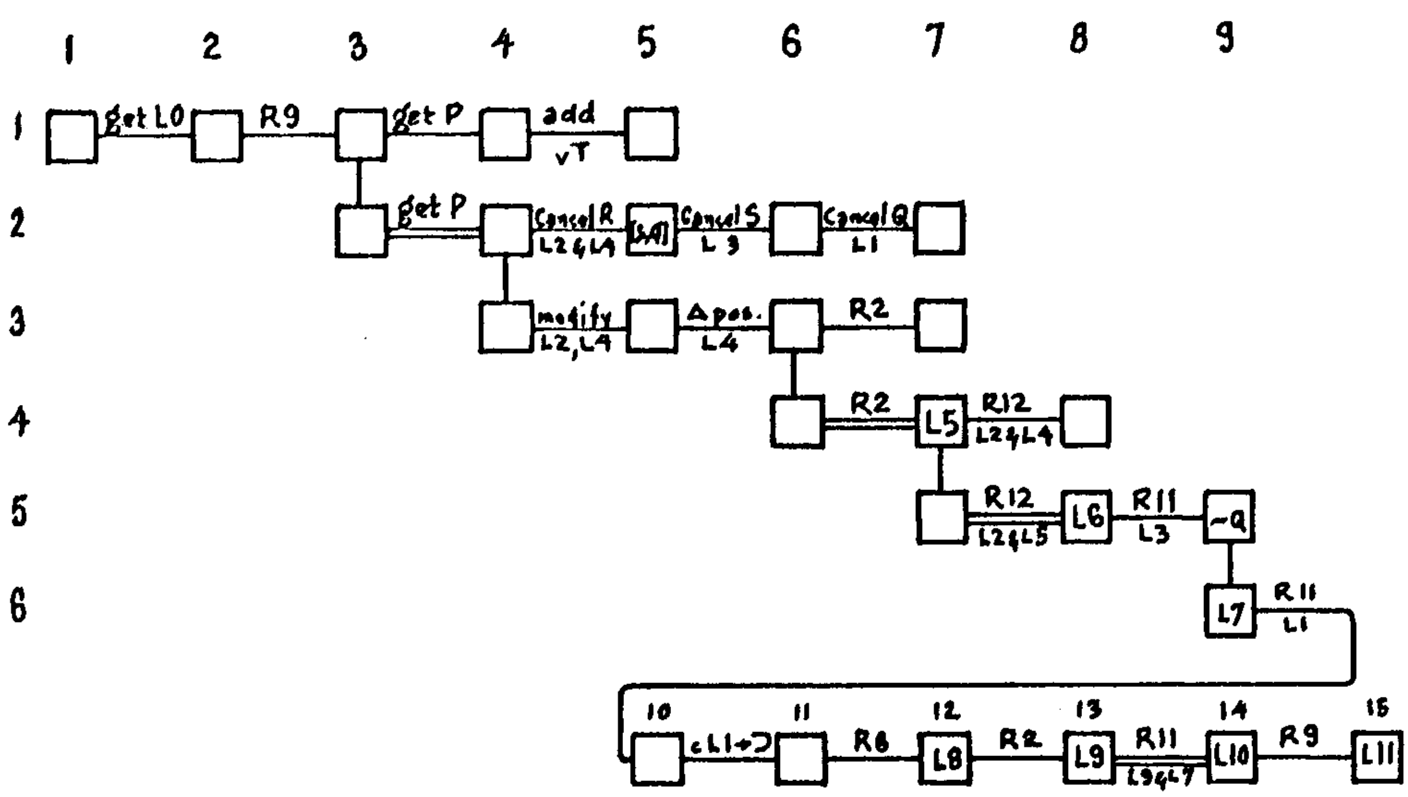
The second modification in the problem space is the use of function terms for operators. The subject not only has the specific rules (R1 to R12), but also "Change sign," "Change connective to wedge," "Delete Q," "Cancel the S's," etc. These expressions play a dual role. First, they are the conversion into an action language of the differences seen between expressions. Given PvQ to be transformed into $-P \supset Q$, the difference in connectives (the v versus the $\supset$) is converted into the statement "Change the connective from wedge to dot," or an abbreviated version such as "Change connective." This is then used to select one of the admissible operators; e.g., Rule 6 in the example above. Thus, function terms play the role of intermediaries in getting from perceptions (differences in characteristics of expressions) to actions (the legal rules). If this was all they did, then they could be absorbed in the process of operator selection and would not appear as operators at all. This is essentially the view taken in GPS, where a table of connections going directly from differences to legal operators was provided.

These function terms become operators at the point where a new state of knowledge is produced as the result of applying a function term, which then becomes the input for another operator, either a legal one or another function term. This happens frequently enough in the various protocols to warrant treating them as operators. Thus, these function terms correspond directly to the function moves in chess (Bishop defends) or the inferences in crypt-arithmetic based on states of knowledge incorporating set information (E even implies E cannot be 9).

Extensive use of function terms as operators constitutes a variety of planning -- of proceeding on the assumption that a sequence of legal operations can be found later that will carry out the transformation implied by the function terms. Figure 16 shows the PBG on a different problem (and a different subject) that leads to an extended plan (lines 1 and 2) with reworking of the plan to fill in the detail (successfully, as it turns out). This form of planning has been analysed elsewhere in more detail using the sorts of goal structures GPS would set up in creating such plans [21].

Missionaries and Cannibals. The missionaries and cannibals puzzle has been used frequently as a task for problem solving programs. Three missionaries and three cannibals wish to cross a river, but have only a boat that holds two people. All can row, but it must never happen that on any shore there are more cannibals than missionaries. The task is to specify the schedule of boat loads back and forth across the river so that all six will eventually end up on the far side of the river.

Figure 17 shows the problem space for a human subject solving the M&C puzzle; Figure 18 shows the PBG [18]. The problem space is again the obvious one: a particular arrangement of missionaries, cannibals and boat being the state of



L1	$P \vee Q$	L0 $P \vee T$
L2	$\sim R \supset \sim Q$	
L3	S	
L4	$R \supset \sim S$	
L5	$S \supset \sim R$	R2 to L4
L6	$S \supset \sim Q$	R12 to L2 & L5
L7	$\sim Q$	R11 to L3 & L6
L8	$\sim P \supset Q$	R6 to L1
L9	$\sim Q \supset P$	R2 to L8
L10	P	R11 to L7 & L9
L11	$P \vee T$	R9 to L10

Subject 8(57)
Logic Problem A4

Figure 16: Logic: Problem Behavior Graph of subject 8 on problem A4.

knowledge, and the various possible boat loads moving across the river being the operators. The one additional feature is that the subject sometimes distinguishes putting men into boat and taking them out as a separate move. This additional elaboration, which is completely non-functional, accounts not only for some of the elaborateness of the PBG, but for some of the blind alleys. In line 6, for example, the subject ignores the constraint on the right because the cannibal doesn't get out of the boat. Again, like crypt-arithmetic and logic, the subject has an external representation, which provides a memory of the current position. In this task actual porcelain figures on a facsimile river were used, rather than a written record, so there was no cumulation of past data, as in logic. Throughout the entire course of problem solving this subject remained within this elementary problem space, except at one point. In line 16, he discovers the crucial move by making two illegal moves in a row. He then combines them legally in line 17. Of course, this does not go outside the problem space, only outside the bounds of strictly legal moves.

Summary. We have now presented PBG's from several tasks. In all cases we get the same information. First, we obtain confirmation that the subject is solving the problem by search in a closed space. Second, we get a characterization of that space in terms of the kinds of knowledge used for states and the kinds of operators for deriving new knowledge. This provides one description of the intellectual level on which the subject is operating. Third, we prepare for the next stage of the analysis -- to ask what can account for the particular search patterns that emerge in the PBG. In some cases, such as chess, we could already generate some hypotheses on the basis of the global features of the graph, without inquiring in detail what choices were made at each point. More generally, if a program were to be constructed to simulate the episode, we would expect it to reproduce the PBG with some fidelity.

States of knowledge:

The configuration on the river, consisting of the location of the boat (▷ on left, ◁ on right) and the location of the missionaries (M) and cannibals (C) on the riverside and in the boat.

Examples:	MMMCCC▷	Initial position: all on left, boat empty
	MMMC.CC▷	All on left, but two C in boat
	MMMC◁CC.	Two C on right, but still in boat
	MMMC◁CC	Two C on right with empty boat
	◁MMMCCC	Final position: all on right, boat empty

Operators:

Moving boat across the river and putting men in and out of boat

Let X be a sequence of M's and C's

→ Move boat from left to right, disembark all men
← Move boat from right to left, disembark all men

→. Move boat from left to right, do not disembark
.← Move boat from right to left, do not disembark

↓X Add X to the boat (note: boat may already have men in it)
↑X Disembark X from the boat (note: may leave some men in boat)

X→ Add X to boat, move from left to right and disembark
←X Add X to boat, move from right to left and disembark

X→. Add X to boat, move from left to right, do not disembark
.←X Add X to boat, move from right to left, do not disembark

Evaluation codes:

-l Too many C on left
-r Too many C on right
c Cycle: return to prior position
i Experimenter interrupts
? Uncertain
\+ Success

Two spaces (both with same knowledge states and operators):

External space: States are squares; operators are solid
Internal space: States are circles; operators are dashed

Figure 17: Missionaries and Cannibals: Definition of problem space

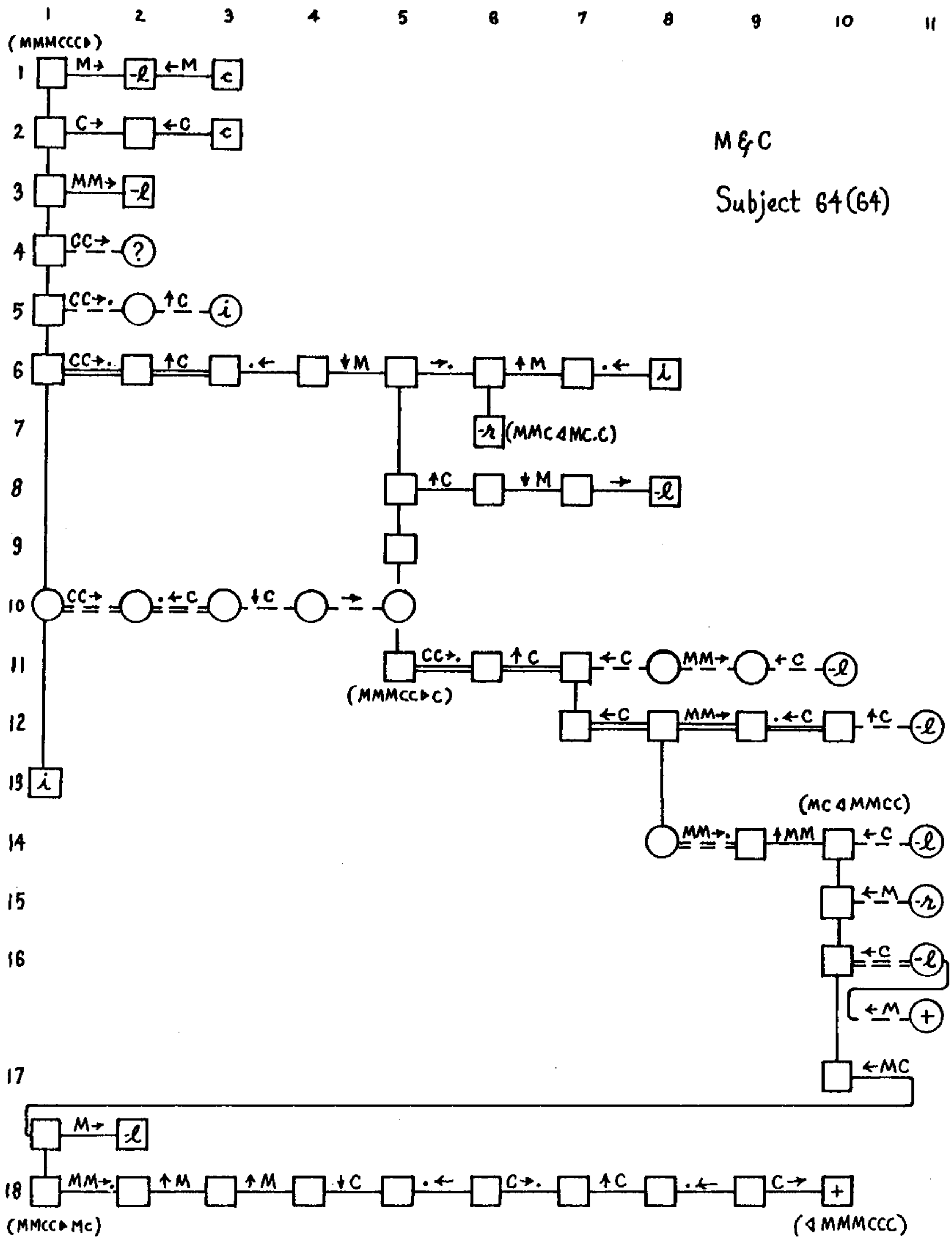


Figure 18: Missionaries and Cannibals: Problem Behavior Graph of subject 64.

We have not discussed the various possibilities for error in creating a PBG, except to comment on the problems of encoding in the crypt-arithmetic case. The problem space permits one to ignore part of what goes on in the protocol, attending only to what indicates a change of knowledge state as defined in the problem space. Thus, much material in the protocol may be left out of consideration. As an extreme example, Figure 19 shows the PBG that would have been generated for the crypt-arithmetic example if one had decided to use the external problem space -- i.e., what was written on the board -- as the state of knowledge; and writing a digit in place of a letter as an operator*. We can see that this graph is much sparser than the graph of Figure 10. One clue as to its inadequacy certainly would be the long stretches of the protocol that lead to no change in state of knowledge. The more important evidence would come, however, from the inability to carry out the next stage of the analysis -- to find any way to characterize the way choices are made in this space.

In general, several kinds of errors are possible in analysing a protocol into a PBG. The problem space might be too aggregated, so that the essential problem solving occurs within a single node of the graph, and the PBG as drawn is concerned only with relatively unimportant features. Alternatively, the problem space might be too detailed, so that the relevant control over search is going on at a higher level, with the steps in the given problem space simply being the means to carrying out these higher level plans. Finally, the problem space might be simply epiphenomenal, so that the real problem solving occurs in some space that does not reveal itself. The clues that indicate each of these errors

---

* 5/D12 means "write 5 at the occurrence of D in column 1, row 2";
3/R2 means "write 3 for the occurrence of R in column 2";
0/T means "write 0 for the occurrence of T."

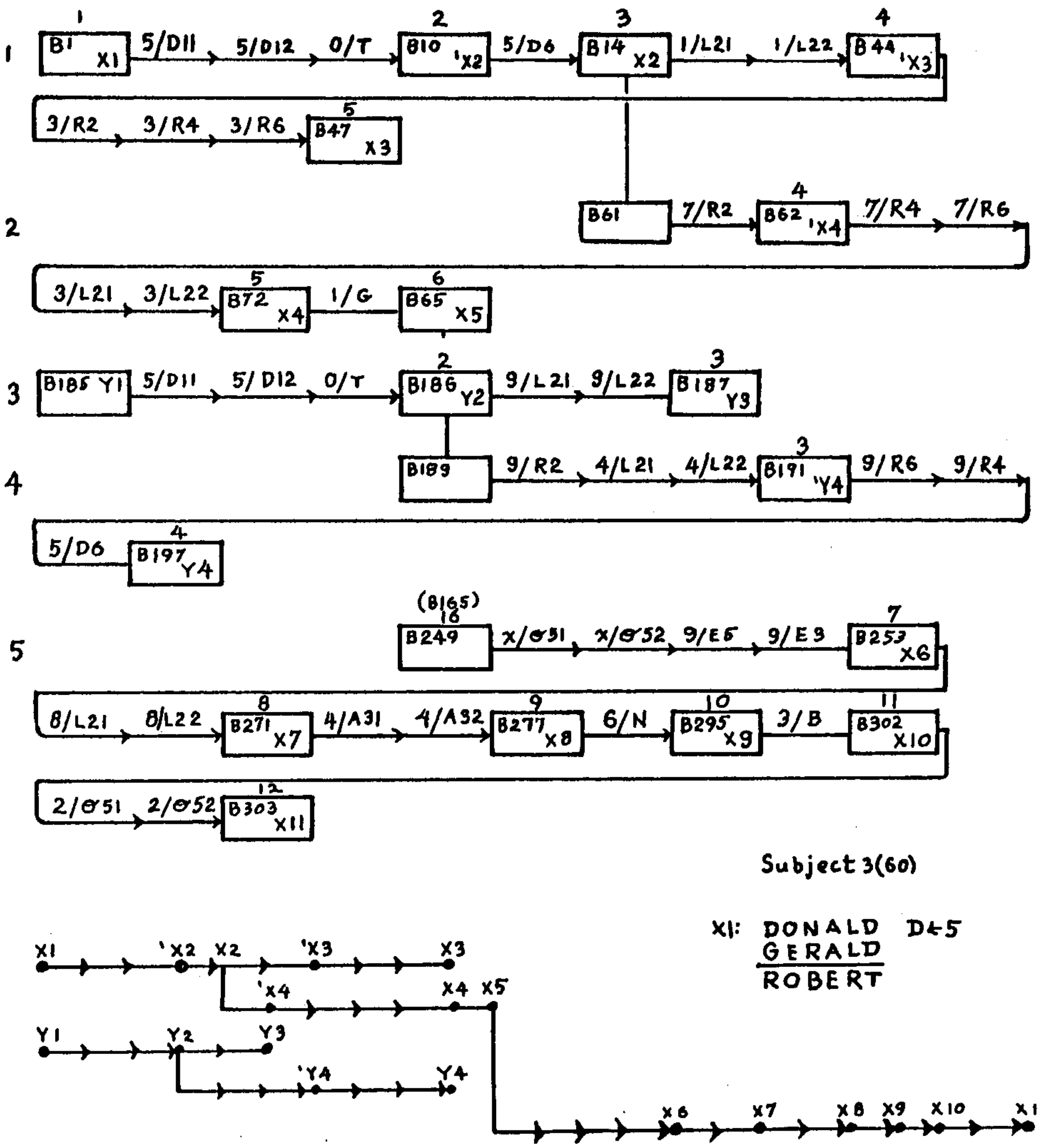


Figure 19: Crypt-arithmetic: Problem Behavior Graph in external problem space.

revolve around the unexplainability of various choices made in the PBG under analysis. Contrariwise, if the problem space is the appropriate one for the episode, then we should be able to describe a collection of processes that collectively perform the functions of Figure 5 in a consistent way.

Analysis of Regularities in the PBG

At each node of a PBG the subject makes a number of decisions (or selections), already summarized in Figure 5. According to the theory, these should be based in large part on the state of knowledge existing at that time; that is, on the state of knowledge associated with the node, including path information and past attempts information. Information outside this state may be used as well, but it is either not covered by the problem space (e.g., time is running out) or is not variable over the course of problem solving (e.g., properties of integers). The subject makes repeated use of these processes of decision, and we get essentially one observation per node of the PBG. This is an "experiment of nature" in that we do not control the population of trials; but if we are lucky we will get a number of decisions in closely related states of knowledge from which we can induce what these decision processes are and whether they are sufficiently stable to replicate themselves.

Production systems. We need some language to express the decision and selection processes that might characterize the subject. We would like a scheme that facilitates inducing these processes, rather than requiring the invention of the complete program all at once. One that appears to have some of the desired virtues is the production system. This consists of a set of productions, each of which consists of a condition expression followed by an action expression:

condition → action

The production is to be considered in the context of the state of knowledge at a node. If the condition is true of the state of knowledge, then the action part is evoked; otherwise the production has no implication for the behavior of the system at that node. In applying a production system (i.e., a set of productions) to a node, some doctrine of conflict resolution is necessary to select a unique action if the condition of more than one production is satisfied. The simplest such scheme is a priority ordering of the productions, so that the one of highest priority always wins out.

Production systems have an extensive history in logic and the theory of algorithms [17]. They have been much used recently in programming, as schemes for handling syntax [6] and doing symbolic computation [9]. Production systems are still a perfectly general scheme for information processing; they simply divide up the computation somewhat differently than a standard sequential programming language. The generality of production systems does not imply theoretical neutrality. They make it easy to express certain forms of organization, hard to express others. Thus, they mold psychological theory to some extent. The issue will not be explored further in this paper, but its existence should be noted.

The advantage of a production system for the task of program induction lies in the fact that at each node one of the productions is evoked. Therefore its condition is true of that state of knowledge and its action occurs at that point. Thus, an hypothesis formed by the analyst at a node takes the form of a proposal for one of the productions that exists in the system. This can be specified independently of what other productions exist in the system. Thus, the total system can be put together piece by piece from a consideration of what happens in each local situation.

The system is not actually as free as the above paragraph indicates. Once a production has been specified, it should be evoked in any situation where its condition is satisfied. Since the states of knowledge are already given in the PBG, the set of nodes where a production is theoretically evoked is determined. Whether it is in fact evoked, as indicated by what action takes place there, is an empirical matter to be answered by an inspection of the PBG. To the extent that the production does occur where predicted, we get confirmation of a regularity in the subject's behavior.

Some extensions to the above picture must be introduced before the scheme for the analysis of regularities is complete. The nodes provide a first segmentation of the protocol. Thus, there will be at least one production per node whose action includes the operator that is evoked at the node. But it is possible to have additional productions whose output is some intermediate information used by another production that leads to the selection of the operator. This intermediate information will not be such as to change the state of knowledge in the problem space, of course. For example, it might be the discovery that all operators had been tried at the node, which would lead to the cessation of the attempt to select an operator and to the evocation of a production leading to the selection of what node to return to. Thus, the total population of observables may increase somewhat as productions are defined.

Secondly, defining the productions locally and in isolation only partially specifies the total production system. Many productions may be predicted to occur at a node. The evidence will indicate which one (or perhaps none) of the predicted set occurred. A conflict resolution rule, such as a priority ordering, needs to be added to complete the production system in a way consistent with the actual occurrences.

A final complication is that we may want to define productions whose action part consists of a sequence of actions to be taken unconditionally. Such a production would cover several nodes. This situation corresponds to the PBG being too disaggregated, so that what is being plotted in the PBG is not a series of independent actions, but the implementation of a more global method.

We are now ready to examine these ideas concretely. We will do this only for the crypt-arithmetic example, and even here we will have to be sketchy, considering how much detail is necessary to describe fully a production system and its coordination with the full protocol. The original analysis [22] provides a fuller account. In the original study of our chess example [27] a partial analysis of this same kind was carried out, which we will not discuss. However, similar analyses are not available for either the logic or the missionaries and cannibals examples.

Crypt-arithmetic. Figure 20 shows the production system for the PBG of Figure 10. The condition part of a production occurs on the left side of the arrow (→) and the action part on the right. The condition is sometimes composite, the bar (|) serving to separate disjunctive alternatives. The underlined letters indicate both variables and the class to which the variables belong, as defined by the problem space. Thus, $\underline{v}$ is a variable which is a letter or a carry. The square brackets are used to identify something or state an additional condition. Thus, in $\underline{e}$[$\underline{v}$] the variable $\underline{v}$ that occurs in $\underline{e}$ is identified; in $\underline{v}$[constrained] only those $\underline{v}$ satisfy the condition that are constrained as given in the subsequent definition. The action part may consist of a sequence of actions (separated by ;). The double arrow (=>) is used to indicate the output of a process.

There are four types of productions. S1 to S5 lead to the selection of an operator of the problem space (PC, GN, AV). In doing so they may require intermediate information about a column, provided either by FC, FA, or GNC, processes that are not operators in the problem space since we decided not to make the column being attended to a state variable.

Selection

S1 v=d*|v←d* → FC(v)=> c; PC(c) (not repeated)

S2 get v|get v=d → FC(v)=> c; PC(c,v)

S3 get l → FA(l)=> c[v]; AV(v); PC(c,l)

S4 get v[constrained]| [simple]=> d[first]; AV(v,d)
l free → GN(v) (not repeated)
[-simple]=> ds; [small] → AV(v)

v constrained = v odd|v even|v>d|v=ds[small]

S5 check cs → GNC(cs)=> c; PC(c)

Goal setting

G1 ee? → get ee (immediately)

G2 ee[v]-p → get v (note: ee-p accepted) (immediately)

G3 check ee[new] → get ee

G4 get ls → FL(ls)=> l; get l

G5 ee! → check ee (not repeated)

Terminating

T1 l=d|GN(l)=> d → TD(l,d)=> { + | l=d-p(ee!) } (not repeated)

T2 ee-p → FA(ee)=> ee'; ee'-p
(except ←)

Repeating

R1 Q=> e[v][unclear] → get v; repeat Q

R2 check ee[old] → FP(ee)=> P; get ee; repeat P

Definitions of additional processes

FC(v) Find a column that involves v. For l, the column includes l, but for t it may be either the carry-out-of column or the carry-into column.

FA(ee) Find the antecedent that generated ee or, if a variable, a relationship that determines v.

GNC(cs) Generate the columns in the set of columns, cs.

FL(ls) Find letter in the set of letters, ls, that is still undetermined and occurs a maximum number of times.

FP(ee) Recall the production, P, that was used to generate the expression ee. (Therefore ee is not a variable.)

Figure 20: Crypt-arithmetic: Production system.

S1 reflects the use of newly achieved information by trying to find someplace where it can yield still other information. S2 is just the opposite; given the goal of getting something, it tries to find a place where something about it can be found out. S3 is an indirect form of assignment; instead of assigning an arbitrary value to _1_ directly, it backs off to something that determines _1_, assigns a value to it and then derives the value of _1_. This tends to assure that one more relationship will be taken into account. S4 is a reaction to obtaining partial information by generating the possible values and assigning one of them as a trial. However, if the generation is complex and there are many of them (more than two), no assignment is made. S4 is the only production with a conditional action sequence. S5 provides for checking an answer by iterating through the columns and adding up each successively; it occurs only once during the course of the protocol.

The second type of production, G1 to G5, leads to setting up a goal, either to get something or to check something. G1 says: if the value of something is unknown, then set up the goal of getting it. This will arise, of course, only in the context where the value of that thing has occurred in some other processing. That is, the knowledge state does not contain an expression, _ee_?, for everything the system does not know. G2 says: if a given statement has been found out not to be true of something, then set up the goal of finding out what is true of that thing. G3 says: one way to check something is to get its value. G4 reduces the goal of getting the members of a set to the goal of getting one of them (the one produced by FL). G5 says: if some fact, _ee_, becomes critically important, as symbolized by _ee_!, then it should be checked. Such items can arise from TD in causing something to be impossible, or from PC.

The third type of production, T1 and T2, is concerned with terminating lines of search. T1 evokes TD, the problem space operator that can declare something not possible; T2 is the backtrack operator that concludes that if something implies an impossibility, then it is, itself, not possible.

The final type, R1 and R2, is concerned with repeating paths already trod. R1 repeats processes that were unclear. R2 says to check an item that has already been produced by some processes, repeat that process. It implies that the subject remembers something about paths already taken, and has this path information accessible as a function of the results produced.

There is not space to discuss fully the psychological implications of this system; they are examined in the more extended treatment. Note that the productions jointly accomplish most of the functions given in Figure 5, but that they are not organized entirely as that figure would suggest. Notice also that the productions are neither novel nor cryptic. Each expresses a meaningful unit of action that is rational at a local level -- that is, adapted to the task at hand. This does not imply that when put together the system adds up to highly rational or effective total behavior. In fact a global judgment on the subject's behavior would be that, although he appeared to know what he was doing, it still took him three to four times as long as it would a really good problem solver.

Given the production system of Figure 20, one can go back to the protocol and determine just what productions occur at each point. The right-hand side of Figure 8 gives a sample of this. In general there is only one production per node, although occasionally more than one, (B8), and sometime a single production covers several nodes (B22.1 and B23). A judgment is clearly involved in whether a particular production occurs or not. However, it is rare for there to be uncertainty between two or more productions. Where it has not been possible to determine what production occurred, either because none of the defined productions fit or because the protocol is too obscure, a question mark (?) has been put down.

Having decided what can be concluded from the protocol about what productions did occur, the next question concerns which productions should have been evoked according to their conditions. (It is not possible for a production to be evoked when it shouldn't, since both condition and action must exist in the data before evocation.) A matrix is obtained, shown in Figure 21*, in which the entry in the i-th row and j-th column gives the number of instances in which both production Pi and Pj should have been evoked, but Pj was in fact evoked. Thus the diagonal entries, (j,j), give the number of times the production Pj occurred in the coding of the protocol. Likewise the sum of the two symmetric entries, (i,j) and (j,i), give the number of times the two productions were brought into competition, so to speak. Their division shows who won. Blanks in the matrix indicate that the two productions never competed, and are to be distinguished from zeros, which indicate competition with no wins.

To finish the specification of the production system a conflict resolution rule is required. We have used a priorty scheme, although it is not entirely satisfactory. Thus, for each pair of productions we want to put higher in the order the one which was chosen most often when there was a choice between the two. That is, put Pj over Pi if the (i,j) entry was greater than the (j,i) entry. If we do this for each entry separately, intransitivities are possible and

* Attend only to the numbers in the upper half of each cell. The figure is reproduced from the more extensive study; the lower number indicate a category of questionable failures, which we do not discuss here.

DID OCCUR

COULD OCCUR

	S1	S2	S3	S4	S5	G1	G2	G3	G4	G5	T1	T2	R1	R2	own ↑	↑	?
S1	35	2	0			3			0		24		3			18	6
S2	8	29	9	15		6	2			4	13		2			11	34
S3	4	16	9	14		3	1			4	9	1	7		16	19	19
S4		2 2	0 2	24		1	0 3		0	1	3	0	1 2		4	4	2 3
S5					1										6		
G1	0	0	0			11			0		0						3
G2		0	0	1			8		0	5		9	11				9
G3					1			2									
G4	26			1		5	4		4	2	18	8	6			3	9
G5		0	0	0			0 1		0	7		4 4	3 2				3
T1	1 2	0	0	0		2 1			0		31		2 2			1 3	2
T2			0	1			5		0	4		10	11				9
R1	0	0	0	0		1	0		0	0 1	0	0	18		18		2 1
R2														4			
own ↑															44		
↑																	
?																	38

Figure 21: Crypt-arithmetic: Matrix showing production conflicts.

indicate that a priority system is not operating. In fact, this procedure leads to a consistent ordering, which is shown in the matrix of Figure 22, where R2 is highest and G4 lowest. The ordering is not fully determined by the matrix. For example, there are no occasions when R1 and S5 were contrasted; consequently they could be permuted in the ordering.

With this priority ordering added, the production system of Figure 20 uniquely determines the production that occurs at each node, except for the ?-nodes. The entries above the diagonal of Figure 22 give the number of errors made by the system in which a production with lower priority occurred in the protocol even though a production with higher priority could have been evoked.

Figure 23 provides a final summary of what the production system has accomplished. It suggests that we could go on adding productions to take care of additional cases in the PBG until -- in the limit -- we would add one production for each node. Thus, we can think of adding productions one by one, getting for each a certain number of cases handled. The main curve, labeled "successful," shows the growth of the total number of situations successfully described. R1, the best production, produces 38 successes; G3, the least successful, adds only 2. Since the productions are reordered according to their successes, we get a smooth curve showing the diminishing marginal utility of the productions in the system.

As noted earlier, adding productions also adds to the total population of observables. This is shown by the curve labeled "relevant," which gives the number of situations in the protocol for which some production (or ?) was coded. The actual total number of situations (275) was slightly higher, since 8 situations were deemed to be clearly outside the problem space and thus should not be counted. An example is B1, which deals with the definition of the rules of the task.

DID OCCUR

COULD OCCUR	R2	S5	G3	G1	R1	T1	G5	S4	T2	G2	S1	S2	S3	G4	own ↑	↑	?	Ex
R2	4																	4
S5		1													6			7
G3		1	2															2
G1				11	0	0		0			0	0	0	0			3	11
R1				1	18	0	0 1	0	0	0	0	0	0	0	18		2 1	36
T1				2 1	2 2	31		0			1 2	0	0	0		1 3	2	31
G5					3 2		7	0	4 4	0 1		0	0	0			3	7
S4				1	1 2	3	1	24	0	0 3		2 2	0 2	0	4	4	2 3	28
T2					11		4	1	10	5			0	0			9	10
G2					11		5	1	9	8		0	0	0			9	8
S1				3	3	24					35	2	0	0		18	6	35
S2				6	2	13	4	15		2	8	29	9			11	34	29
S3				3	7	9	4	14	1	1	4	16	9		16	19	19	25
G4				5	6	18	2	1	8	4	26			4		3	9	4
own ↑															44			
↑																		
?																	38	

Figure 22: Crypt-arithmetic: Reordered matrix according to priority rule.

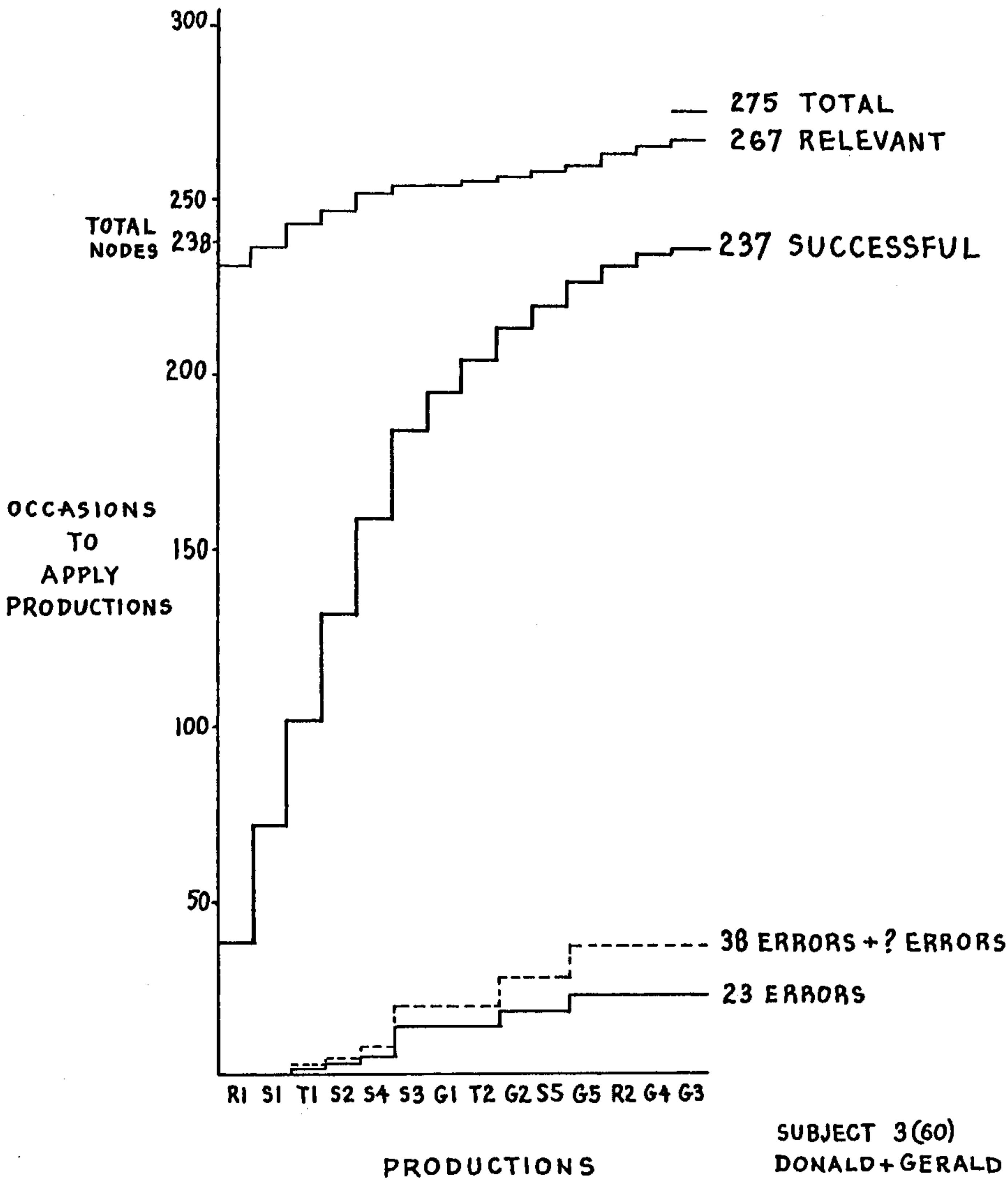


Figure 23: Crypt-arithmetic: Summary of performance of production system.

As productions are added, the number of cases not handled (?-nodes) diminishes. At the same time errors arising from conflicts between productions gradually increase. This is shown by the solid lower curve, labeled "errors," which starts at zero and climbs eventually to 23*. This comes from summing the appropriate entries above the diagonal in the matrix of Figure 22.

Let us summarize where we have come. Technically any production system is a program. If presented with an initial information state,it will evoke a sequence of productions, executing their actions and modifying the information state accordingly. This will continue until either the system loops, or a state is reached where no production is evoked, at which point the system stops. The system of Figure 20, although a program in this sense, is not yet a full program either for simulating the subject or even for solving crypt-arithmetic tasks. If set loose on a new task,

```
 SEND
 MORE
MONEY
```

it would not know what to do with the left hand column, which has blanks in it. That is, the productions have all been built around one episode and have not been extended to form a complete system. They have not even been extended to cover the ?-nodes, so that as a system it will not keep going for the DONALD+GERALD task. Instead, Figure 20 represents the regularities found in the protocol and has expressed them in a form in which any program that is built can take them into account. Such final programs can be made by extending the system by additional productions until it is complete, say, over all crypt-arithmetic tasks of the simple type used here. Indeed, this seems the natural

* The dotted curve, labeled "errors + ?-errors," adds to the error curve the additional entries from the lower half of each cell of the matrix. As noted in the earlier footnote, we do not have space to discuss the nature of these "questionable" errors.

way to proceed. But one could also proceed using more conventionally organized programs with a more constrained flow of control, or trying to embed the process into a structure such as GPS. In these latter cases, the production system, along with the summary of how well the various productions fared, provides strong statements about what has to go into the simulating system.

The matrix of Figure 22 and the accounting of Figure 23 also suggest we may have made some progress on the assessment of an information processing theory. We have managed to obtain an ensemble of instances and to divide our process representation up into pieces that can be handled as individual units, so that we can count successes and failures. It is still unclear what these counts mean in the sense of any underlying statistical theory of the expectation of various degrees of success. But it is already clear that empirical norms are possible. For instance, the appropriate way to record the present venture might be as a system of 14 productions in a task with a population of 267 evocations yielding a coverage of 85% with conflict errors of 10%. This could be compared with behavior on other crypt-arithmetic tasks, and even with behavior on other tasks. A population of such figures might serve to indicate whether a proposed theory in fact yields an improvement and in what way. Such information would be exceedingly useful, even without any formal theory of significance*.

* It has been suggested [31] that one might be able to use the kind of information transmission analysis described by Garner [10]. The productions would be viewed as reducing the amount of uncertainty one had about the data, and under suitable assumptions one might calculate a specific figure for this. As of now, it is unclear to me what such further aggregations would add to the summaries of Figures 22 and 23.

Summary

Let us pull together the threads of the story. We have been concerned with making protocol analysis into a useful tool. This has led to a methodological emphasis with, however, the focus on improving the technology for developing theory, rather than for validating theory. We introduced a series of steps in the data analysis whose function was to make evident the important regularities in the protocol, and pave the way for constructing process models of the subject's behavior. Briefly summarized, these steps are:

Divide the protocol into phrases. Each phrase represents a single assertion about the task or a single act of task oriented behavior. Although trivial, this step is worth noting, since it represents the limit of precoding of the verbal behavior.

Construct a problem space. Both the operators and the information constituting a state of knowledge are set down. There may be more than one problem space, of course. The problem space is a hypothesis about the subject's behavior.

Plot the Problem Behavior Graph (PBG). Proceed through the protocol phrase by phrase. The key constraint is that all changes in knowledge state (as defined for the problem space) that are detectable in the protocol must come about through application of one of the operators of the problem space. The PBG segments the protocol into a population of occasions for action.

Create a production system. This system attempts to capture the regularities in the search behavior. It can be viewed (with some literary license) as proceeding in several steps:

<u>Conjecture individual productions</u>. At each node of the PBG conjecture a production that responds to features in the knowledge of that node (essentially known through the construction of the PBG) and yields the action taken. This leads to a large collection of individual productions.

<u>Consolidate the production system</u>. Rewrite as many productions as possible as variations on a few, thus reducing the total number of productions in the system. This is analogous to subroutinizing a large program, and yields the same dividends in permitting the essential organization of the system to emerge.

<u>Plot the production system against the PBG</u>. Proceed through the PBG node by node. For each determine not only what production occurred, but what others could have occurred, but didn't.

<u>Determine a conflict resolution rule.</u> This may be a simple priority system, as used here, but it may involve quite different distinctions. For example, it may lead to elaborating the conditions of some of the productions. The matrix of Figure 21 showing how productions fare in competition with each other is a useful display.

This analysis scheme is still incomplete, as we have not carried it through the final steps of getting a running program. These latter steps are not superfluous. They provide the verification that we have a sufficient set of processes for carrying out not only the immediately present task, but others of similar character as well. In addition, the hand codings engaged in during the preliminary steps described in this paper always leave something to be desired by way of accuracy. The final system as a running program provides much stronger quarantees.

In our emphasis on the methodology, we have slighted the psychology.

As already noted, production systems carry additional psychological implications beyond those already apparent in the problem solving theory we laid out explicity. We have not discussed these, nor have we discussed the nature of the particular production system we derived. Finally, even assuming we accept a production system as an appropriate way to express the microtheories, we have not explored how these contribute to the more general information processing theory of problem solving.

A final note should be made about the scope of the techniques. Although it is reasonably clear that they apply to tasks involving the exploration of consequences, it is unclear how far they stretch. For example, no evidence is available yet for concept formation tasks, even though some of these have made good use of protocols [7, 13, 16].

References

1. R. F. Abelson, "Computer simulation of 'hot' cognition," in S. S. Tomkins and S. Messick (eds.) Computer Simulation of Personality, New York: Wiley, 1963.

2. M. Brooke, 150 Puzzles in Crypt-arithmetic, New York: Dover, 1963.

3. K. M. Colby and J. P. Gilbert, "Programming a computer model of neurosis," J. Mathematical Psychology, vol. 1, no. 2, July 1964, 405-417.

4. A.D.DeGroot, Thought and Choice in Chess, The Hague: Mouton, 1965.

5. K. Duncker, "On problem solving," Psych. Monogr., Whole No. 270, 1945.

6. A. Evans, "An ALGOL compiler," Annual Review of Automatic Programming, vol. 4, 1964, 87-124.

7. J. Feldman, "Simulation of behavior in the binary choice situation," in E. Feigenbaum and J. Feldman, (eds.) Computers and Thought, New York: McGraw-Hill, 1963.

8. J. Feldman, F. Tonge, and H. Kantor, "Empirical explorations of a hypothesis-testing model of binary choice behavior," in A. Hoggatt and F. Balderston (eds.), Symposium on Simulation Models, Cincinnati: South-Western, 1963.

9. R. Floyd, (ed.) Proc. ACM Symposium on Symbolic and Algebraic Manipulation, published as a special issue of Communications of the ACM, vol. 9, no. 8, August, 1966

10. W. R. Garner, Uncertainty and Structure as Psychological Concepts, New York: Wiley, 1962.

11. H. Gelernter, "Realization of a geometry-theorem proving machine," in E. Feigenbaum and J. Feldman (eds.), Computers and Thought, New York: McGraw-Hill, 1963.

12. H. Gelernter, "Machine generated problem-solving graphs," in Proc. of the Symposium on Mathematical Theory of Automata, New York: Polytechnic Press, 1963.

13. L. Gregg, "Internal representations of sequential concepts," in B. Kleinmuntz (ed.) Concepts and the Structure of Memory, New York: Wiley, 1967 (in Press).

14. J. R. Hayes, "Memory, goals and problem solving," in B. Kleinmuntz (ed.), Problem Solving: Research, Method and Theory, New York: Wiley, 1966.

15. G. Humphrey, Thinking, New York: Wiley, 1951.

16. E. S. Johnson, "An information processing model of one kind of problem solving," Psych. Monogr. Whole No. 581, 1964.

17. A. A. Markov, Theory of Algorithms, Academy of Sciences, USSR, 1954.

18. W. McPhee, Unpublished data, 1965.

19. G. Miller, E. Galanter, and K. Pribram, Plans and the Structure of Behavior, New York: Holt, 1960.

20. M. Minsky, "Steps toward artificial intelligence," in E. Feigenbaum and J. Feldman (eds.) Computers and Thought, New York: McGraw-Hill, 1963.

21. A. Newell, "The possibility of planning languages," in F. Geldard (ed.) Communication Processes, New York: Pergamon, 1965.

22. A. Newell, Studies in Problem Solving: Subject 3 on the Crypt-arithmetic Task DONALD+GERALD=ROBERT, Carnegie Institute of Technology, July, 1966.

23. A. Newell and G. Ernst, "The search for generality," in W. Kalenich (ed.) Proceedings of IFIP Congress 65, vol. 1, Washington: Spartan, 1965, 17-24.

24. A. Newell and H. A. Simon, "GPS, a program that simulates human thought," in E. Feigenbaum and J. Feldman (eds.) Computers and Thought, New York: McGraw-Hill, 1963.

25. A. Newell and H. A. Simon, "Information processing in computer and man," American Scientist, vol. 52, no. 3, 1964, 281-200. (see the appendix)

26. A. Newell and H. A. Simon, "Programs as theories of higher mental processes," in R. W. Stacy and B. Waxman (eds.) Computers in Biomedical Research, vol. 2, New York: Academic Press, 1965.

27. A. Newell and H. A. Simon, "An example of human chess play in the light of chess playing programs," in N. Weiner and J. P. Schade (eds.) Progress in Biocybernetics, vol. 2, Amsterdam: Elsevier, 1965.

28. A. Newell, J. C. Shaw and H. A. Simon, "Empirical explorations with the Logic Theory Machine," in E. Feigenbaum and J. Feldman (eds.), Computers and Thought, New York: McGraw-Hill, 1963.

29. J. J. Paige and H. A. Simon, "Cognitive processes in solving algebra word problems," in B. Kleinmuntz (ed.), Problem Solving: Research, Method and Theory, New York: Wiley, 1966.

30. W. R. Reitman, Cognition and Thought, New York: Wiley, 1965.

31. W. R. Reitman, personal communication.

32. H. A. Simon and E. Feigenbaum, "An information processing theory of some effects of similarity, familiarity and meaningfulness in verbal learning," J. Verbal Learning and Verbal Behavior, vol. 3, no. 5, 1964, 385-396.

33. L. Uhr, C. Vossler and J. Uleman, "Pattern recognition over distortions by human subjects and by a computer simulation of a model for human perception," J. Experimental Psychology, vol. 63, 227-234, 1962.