PROTOCOL ANALYSIS AS A TASK FOR ARTIFICIAL INTELLIGENCE

D. A. Waterman and A. Newell

May 24, 1971

This paper is to be given at the Second International Joint Conference on Artificial Intelligence, London, September 1-3, 1971. It may not be reproduced without the permission of the authors. This research has been supported primarily by Research Grant MH-07722 from the National Institute of Mental Health and in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44610-70-C-0107) and is monitored by the Air Force Office of Scientific Research.

Departments of Psychology and Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania

CIP Working #178

PROTOCOL ANALYSIS AS A TASK FOR ARTIFICIAL INTELLIGENCE

D. A. Waterman and A. Newell

Departments of Psychology and Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania

Abstract

We are attempting to automate protocol analysis, which is a form of data analysis in psychology for inferring the information processes used by a human from his verbal behavior while solving a problem. The paper discusses protocol analysis as a task in artificial intelligence. The discussion is based on (but broader than) our current program, PAS-I, which creates a description of a subject's changing knowledge state from his verbal behavior.

I. Introduction

A form of data analysis, called protocol analysis, has been much used in recent work in the psychology of thinking and problem solving. The subject talks while attempting to solve a problem, his verbalizations are transcribed, and the underlying information processes are inferred from their content. Examples of tasks subjected to protocol analysis are various puzzles, such as Missionaries and Cannibals (4) or cryptarithmetic (12, 13, 17, 26) elementary logic problems (14, 15), chess (6, 7, 16), binary-choice sequence prediction (9), geometry proofs (4), word problems in elementary algebra (20), concept identification for the induction of various logical and sequential concepts (19), and various understanding tasks (21).

Our long-term goal is to automate protocol analysis. Careful protocol analysis is time-consuming, and extensive analyses requires automatization. A considerable increase in objectivity may occur, since the analysis will be accomplished with determinate rules, rather than by a human with indeterminate intellectual powers. Finally, an explicit representation may be possible of the evidence provided by a protocol for or against a given theory of human problem solving.

Two side interests are served by this project. First, the task to be automated -- the analysis of protocols -- requires an artificial intelligence program, since the functions involved include extraction of meaning, inference from data, and induction of new sets of rules. Second, since understanding the content of freely produced natural language is central to protocol analysis, the results may be of interest to those concerned with semantics.

We currently have running an initial system for automatic protocol analysis, called Protocol Analysis System I (PAS-I), designed to handle protocols for the task of cryptarithmetic. A complete description of the program with examination of its behavior in some detail is the subject of a companion paper to be presented to a psychological audience (25). The present paper examines protocol analysis as a task for artificial intelligence -- the essential problems, the task representations, and the methods. It draws extensively on our early experience with PAS-I, but goes beyond it at several points.

II. Methodological Preliminaries

Automating protocol analysis is a long-term effort involving many difficulties. This puts a premium on adopting a sensible strategy for carrying out the project. We describe here some of our cardinal tenets.

First, the system is primarily for our own use. We ourselves are involved in studying cognitive processes and analyzing protocols. We expect others to use automatic protocol analysis techniques when they are developed; but adaptation to the needs of others is a postponable task.

Second, initial attempts at a difficult task should focus on a specific variant. Generality can come later. Thus, we have picked a specific problem solving situation, cryptarithmetic, and ignored all others, such as chess, logic, concept identification, etc. The selection of cryptarithmetic is based on the relatively sophisticated and successful development of a particular style of protocol analysis for this task in prior work. Success with cryptarithmetic could lead to rapid scientific gains in terms of questions already posed in this area that cannot be explored without extensive analysis of many protocols. Consequently, this specialization may provide an early justification of the work, even without solving any of the problems of generalization that clearly lie just beyond.

Third, developing complex programs is an experimental activity. The touted procedure of careful planning, followed by complete specifications prior to coding, is exactly the wrong way to proceed. Every component of the system will be redesigned and recoded not once but many times. The important step is to get a version of the program written and running, to obtain feedback for the next iteration. Thus, the current set of design decisions in PAS-I do not represent conceptual commitments on how the task should be done, but simply our current selection of mechanisms to try. This system uses SNOBOL4 for the linguistic front end and LISP for the analytical back end -- clearly a temporary expedient.

Fourth, complex software systems should be designed and built by very few people (here two), a principle much quoted in computer science. For artificial intelligence systems of moderate size, we think this principle is actually feasible. It does appear essential for experimental programming.

Fifth, one should aim at full automatization and not at some optimal man-machine symbiotic system, even though the latter is the desired goal. Selection of a man-machine system as the top-level goal invariably puts strong emphases on the division of labor between man and machine and on the hardware and software for communication. Both of these aspects seem secondary in importance, especially in a long-term development. Moreover, posing the design problem as the optimal division of labor encourages attitudes like "the man should do what requires creativity and intelligence; the machine should do what requires drudgery and repetitive calculation." These distort the design and are ultimately self-limiting in terms of preconceived notions of the powers and limitations of both computers and men. We prefer to devote our efforts to automating the central intellectual functions involved in protocol analysis. Adaptation to an appropriate man-machine system is then a secondary effort.

III. Framing the Problem

Protocol analysis, as it currently stands, is an informal art, where each investigator uses materials in ways that suit his needs. The work in cryptarithmetic (13, 17) constitutes a refined form of protocol analysis, involving a definite series of data analytical steps and considerable detail of the verbal utterances. We follow the general scheme of this analysis, though it constitutes a substantial narrowing of the task.

The experimental situation is fixed. The subject is given a problem by means of instructions as shown in Figure 1. A tape recording is made of his utterances throughout the session. Note is taken of each act of writing and its time, so coordination is possible with the speech. This audio tape constitutes the primary data to be analyzed. Figure 2 gives a transcription of the tape for the initial part of a session analyzed previously (13) (called S3's session).

The final output of an analysis is a description of the subject as an information processing system. It consists of two structures. The first structure is the problem space, which specifies the kinds of knowledge the subject can have about the task -- what he can come to know. This can be done in a grammar-like way by giving a language. Any expression in this language represents a possible state of knowledge of the subject, hence a possible point in the problem space. Included in the notion of a problem space are the means to obtain new information from old: a finite set of operators which take a state of knowledge as input and produce a new state of knowledge as output. These operators are incremental, adding or modifying only a small part of the total knowledge state. Figure 3 shows a simplified version of the problem space for S3, using BNF.*

At the top of Figure 3 are the entities about which something can be known. Below this are seven expressions, e.g., (EQ D 5) says the subject knows that D is 5. The knowledge state is the conjunction of a number of such expressions. At the bottom are the four operators by which the subject can produce new knowledge.

The second structure is a production system (similar to Post or Floyd productions), consisting of an ordered set of productions. Each production consists of a condition part and an action part, conventionally written as:

$$\text{condition} \rightarrow \text{action} .$$

The condition part consists of tests that can be applied to states of knowledge, as given by the problem space. The action part consists of a sequence of one or more operators. A production system can be applied to a state of knowledge by executing the action of the first production (in an ordered list) whose condition is true of the knowledge state.** A production

* The notation in Figures 3-6 has been changed from the original paper (13) to conform with that used in PAS-I.

** If the action is a sequence of N operators then a corresponding trajectory through N nodes of problem space is generated by a single production. Without loss of generality actions could be limited to a single operator.

system forms a complete process if it is iteratively applied to each new knowledge state that is generated by its actions. Figure 4 gives a simplified illustrative fragment of the production system for S3. Production P1, for instance, has a condition that is satisfied by expressions such as (EQ R 7) or (AEQ L 1). If the condition is satisfied, then two operators are applied. The first, FC, selects a column to work on; the second, PC, processes that column to obtain new knowledge. The production system requires some additional operators, not in the problem space of Figure 3. These operators (FC and FL) obtain the operands for the main problem space operators, rather than obtaining new knowledge about the task.

Besides these two static structures, which constitute the model of the subject, the analysis also provides two dynamic representations of the subject's behavior. The first, called the Problem Behavior Graph (PBG), describes the trajectory of the subject through the problem space. Each node of the graph represents a particular state of knowledge and each branch represents the operator that was applied at that state. Since the subject may return to the same state of knowledge at different times, the graph is conventionally drawn with a distinct node for each distinct visit to a knowledge state. Thus, conventionally time runs across the page from left to right and then down. Figure 5 shows a simplified problem behavior graph for the initial part of the session of S3. The knowledge states are represented by the nodes (square boxes) and the application of the operators by the branches. Comparison with Figure 2 will show that some actions are not represented explicitly, e.g., writing results (at lines 8, 9, 12, and 13) and obtaining a letter to work on (lines 14-19). S3 processes column 2 twice (lines 22 and 25) and this is shown as a back-up in the PBG.

The second dynamic representation is the trace of the behavior of the production system, which shows the sequence of knowledge states that the production system generates in attempting to model the subject's behavior. Figure 6 shows the initial part of the trace from the illustrative production system of Figure 5. Both the production and the operator being evoked are given at the left. The next line below gives the output, which can be an intermediate result (such as the column found by FC) and a new addition to the knowledge state. The trace does not carry through the back-up of Figure 5, since additional productions are required beyond the fragment in Figure 4 to recognize the need for repeating and to accomplish it.

These two representations, the trace and the PBG, provide the primary means of assessing the adequacy of the model of the subject, as given by the problem space and the production system. Various measures can be taken on them to summarize the degree of correspondence and to pinpoint the aspects that are especially well accounted for or that create important difficulties.

As stated, these constructs may seem arbitrarily imposed. In fact, they derive from a particular theory of human problem solving. This theory has been expounded at length in Newell and Simon (17)* and there is no need to redescribe it here. We will take these four structures, illustrated in Figures 3-6, as the required outputs of a protocol analysis.

The boundary conditions of the task of protocol analysis are now fixed, with the audio tape on one end and the four structures that make up the psychological model at the other end. Within this domain, however, are many subtasks: description, prediction, induction, evaluation, etc. Each offers its own challenge as an effort in artificial intelligence, though all are ultimately intertwined.

The diversity of subtasks within protocol analysis is compounded by the necessity of several intermediate representations between the tape and the psychological models. Current knowledge is simply not organized for direct transformation between the two. In fact, to proceed further in delineating protocol analysis we must propose a concrete set of these intermediate representations. Figure 7 shows our current set. This is a critical step, for it fixes much of the analysis. These representations are determined primarily by the form of current knowledge. Either we conform to the representations in which a given source (e.g., linguistic knowledge) is expressed or we cannot use the knowledge. Conceivably knowledge could be reworked into some new representation, but this is quite difficult. Thus, we settle for conventional representations and a conventional decomposition of the task.

The first intermediate representations are linguistic ones, involving phonemes, words, phrases, and sentences. The two types of linguistic representations currently employed are shown in Figure 7. The lexical representation consists of the stream of words uttered by the subject, including word fragments, prosodic features, timing information, and paralinguistic features. It is the typical output

* The theory is an outgrowth of work over more than a decade (18). For earlier versions of the theory as it will be used here, see (12, 13, 16). Also a brief summary is included in the companion paper (25).

produced by a human transcriptionist from the audio tape (see Figure 2). The second linguistic representation is the topic representation. This is a segmentation of the lexical representation into units called topic segments, each concerned with a single task topic. In Figure 2 each numbered line is a topic segment. Other linguistic representations are possible (e.g., one into sentences based on a grammatical analysis). We also indicate in Figure 7 that linguistic rules are a necessary source of knowledge in order to work with any of the linguistic representations of behavior. These rules are based primarily on conventional linguistic knowledge (as contained in grammars and lexicons), but also have a component that is idiosyncratic to the subject as well as one related to conversational rules.

The next representations are called semantic ones. They hold the task-related meaning to be extracted from the linguistic representations. They consist of a set of semantic elements, each of which makes an assertion about the experimental situation at some time. The elements fall into two classes. The first, called problem-space elements, asserts the occurrence of some basic item in the problem space, either knowledge the subject has (called a knowledge element) or the occurrence of an operator (called an operator element). The second class, called indicator elements, asserts relations between various elements of the problem space, e.g., that a given knowledge is an input to a given occurrence of an operator. Table 1 gives brief descriptions of the semantic elements currently in use. For brevity, we will drop the word element, when the context is clear, and simply refer to knowledge and operators.

The semantic elements can be arranged as functional units or groups. The operator group consists of an operator along with the knowledge it uses (its input) and the new knowledge it produces (its output). The protogroup is a conjecture of an operator group, formed at an early stage in the analysis.

The next representations are the ones of psychological significance: the PBG (problem behavior graph) and the trace of the protocol system. In terms of the semantic elements just defined, the nodes of the PBG are operator groups. Besides the two behavioral representations (the PBG and the trace) there are two structural representations: the problem space and the production system. It can be seen from Figure 7 that the problem space is necessary to define the elements at the semantic level.

Finally, there are various representations which we have called assessment representations. These are of little interest here, being primarily the results of measurement and statistical algorithms executed on the appropriate basic structures (PBG, trace, problem space and production system).*

The various subtasks encompassed by protocol analysis can be defined in terms of the representations in Figure 7. They arise from the many ways one can obtain information expressed in a particular representation, when given the information in other representations. Figure 8 lists seven broad categories of the subtasks, which run the gamut of recognizable scientific activity. Additional variations can be defined easily.

In the form in which they arise in protocol analysis these subtasks are all specific enough not to have been dealt with directly in the artificial intelligence literature. The work that seems most related are those usually classified as inductive programs. The work on Dendral (5) is by far the closest, since it too deals with problems of inference in an actual scientific context (the structure of organic molecules). The inductive problems usually dealt with (8, 10, 11, 22) are taken in the main from formal puzzles. They seem somewhat remote, though their general lessons about creating spaces of hypotheses are quite relevant. Work by Amarel (1, 2, 3) on inducing functions from input-output tables is also relevant to one class of induction problems that arises here. More generally, Amarel has attempted to outline a class of theory formation problems which would cover a number of the types described here. Work on language, not only linguistic theory and computational linguistics, but also work on semantics and on programs to understand linguistics, is also relevant.

These subtasks do not each require an independent approach and an independent program, as they are defined with respect to the same representations and sources of knowledge. Neither can they be developed all at once. We have started with the problem of behavior description. PAS-I finds the PBG from the topic representation, given the linguistic rules and the problem space. As will be seen, this task is not merely "descriptive," but involves inferring meaning from a sequence of words. It also involves inferring the current knowledge state of a human, given that some past knowledge may have been discarded.

PAS-I constitutes our current state of technical accomplishment, and we will comment on it in some detail. However, the purpose of the

* However, representing the total evidence a protocol offers for a given problem space is an unsolved representational problem.

paper is to describe the larger task of protocol analysis; PAS-I simply tackles one component task. Thus, we will discuss the problem of describing behavior starting with the pure lexical representation (i.e., before segmentation into topics). We will also discuss the description of behavior beyond the PBG to the trace of the production system.

The remaining behavior description problem is the recognition of speech -- going from the audio tape to a lexical representation. Although we will not discuss the problem here, it must be included within the scope of protocol analysis. The evidence from current work in speech recognition implies that the recognition process makes use of linguistic, semantic, and task information. Thus, significant feedback exists from the levels of analysis we do deal with (Figure 7) to the input data associated with these levels.

Of the other tasks in Figure 8 we will discuss here only induction. Current manual analyses of protocols have not moved much beyond descriptions of behavior and induction of the various static structures. Indeed, making protocol analysis easier to do appears to be a precondition to tackling these other tasks.

IV. Description of Behavior: PAS-I

PAS-I takes as input a linguistic representation in terms of topic segments, i.e., groups of words dealing with a single task focus, and delivers as output the PBG. Both the problem space and the linguistic rules are taken as given (the production system is not involved). The problem space is that used by most adults with a Western, moderately technical education, the so-called augmented problem space (17).

Figure 9 shows the overall flow diagram for PAS-I. The first stage consists of a transformation from a linguistic representation (the topic segments) into a set of semantic elements. In the second stage these elements are processed and refined to produce tentative groupings of elements. The third stage involves processing these groupings, refining them further by means of inferential techniques to produce groups consisting of one operator element and its associated input and output knowledge elements. In the final stage these groups of elements are incorporated into the PBG. Feedback exists between the last two stages. The inference processes (determining unknowns and finding origins of knowledge) make strong use of the knowledge state of the subject. Consequently, the PBG must be recomputed with every change of knowledge, so it can provide an accurate estimate of current knowledge. As a result, processing does not proceed in a pipeline fashion in which each representation is computed completely on the basis of lower level information.*

The feedback loop emphasizes a general principle: that information at any level can be brought to bear to determine a particular item. Thus, the separate intermediate representations do not have validity independent of the total analysis. Extensive use of feedback indicates a breadth-first, parallel scheme of computation. But matters will not remain even this simple and subsequent versions will use data not yet processed to help analyze the data currently being processed.

The Linguistic Processor

Figure 10 illustrates the operation of the initial stage, the Linguistic Processor, in more detail. A single topic segment is handled at a time. It is processed by a grammar to yield a set of semantic elements. This grammar is philosophically a key-word grammar that responds directly to cues for the occurrence of the various elements.

Each example of Figure 10 shows the topic segment, its analysis in terms of linguistic classes, and the final semantic elements produced. Figure 11 gives (in a modified BNF notation**) the fragment of the grammar needed to process the examples of Figure 10. These represent only a small part of the rules used by the Linguistic Processor (see the companion paper for the complete grammar and a detailed description of its use). Notice that often more than one element can be produced from a single segment. The segments usually reflect a single topic, yielding one problem space element, plus possibly some related indicator elements. But, as example (f) shows, the grammar does not depend absolutely on there being only one topic per segment and can generate two independent elements. The ability of the grammar to do this is relatively weak, and the assumption that the sequence of words reflects a single topic is strongly built in.

* Currently, the first two stages do not depend on feedback and can be produced on separate passes. Later versions of PAS, however, will incorporate feedback to all stages.

** Here a vertical bar (|) indicates disjunction, and the absence of a blank indicates concatenation, e.g., <a> := B C D | EF defines the class a, consisting of all expressions containing B, C, and D, in that order, or containing EF. Thus BCD, EF, BCAD, BRCLD, and QBSSCRDA are all members of class a. The null string is represented by < >.

An important feature of the Linguistic Processor is its avoidance of a standard grammatical analysis. No irrevocable commitment is implied thereby, though we are disposed to explore such a strategy thoroughly. Language is highly overdetermined; the meaning of a sentence can be inferred from many partial aspects: syntactic, semantic, paralinguistic, and contextual. An extremely strong semantic component is available in the problem solving theory for cryptarithmetic, as represented in the problem space. Thus, it seems appropriate to see how far semantic analysis can carry us. Actually, grammars are not available for the sort of fragmented and ungrammatical speech with which we have to deal, though the departures from full grammaticality do not seem insuperable.

To show the limits of the present analysis, Figure 12 lists several examples of some of the more complicated types of fragmented and ungrammatical utterances the Linguistic Processor accepts as input. Those segments for which the linguistic analysis is clearly inadequate and where no improvement in the keyword type grammar appears likely to suffice (outside of including the segment itself as a special case) are marked with asterisks. In the unmarked examples, however, enough task information was extracted to enable the rest of the system to provide an adequate analysis.

The grammar is given; i.e., it is not determined by the analysis. It is, however, based on several kinds of knowledge. Basic grammar and dictionary knowledge in some way enters throughout. There is considerable special usage due to the task definition, e.g., the use of letters and digits and the relevance of terms such as "writing" and "column at the left." Though these words retain their normal English usage, they are in the grammar only because of the particular task and its physical arrangement. Beyond the task definition is the problem space. Certain arithmetic concepts, such as "even" and "odd" would not be included for a subject who did not use the augmented problem space. Thus, it appears that the linguistic rules are not independent of the other structures posited in Figure 7. Finally, the subject sometimes chooses uncommon ways of saying things. In a limited grammar, it may be necessary to consider the uncommon ways as idiosyncratic to a subject.

The Semantic Processor

The second stage of PAS-I is the Semantic Processor. Here a stream of linguistically derived semantic elements is arranged into initial approximations of operator groups, each containing an operator element and the surrounding knowledge and indicator elements. We call them protogroups, to emphasize the substantial inferential gap between these initial groupings and the final operator groups that are input to the PBG.

Actually forming the protogroups is the last step in a three-step process illustrated in Figure 13. The first of these steps does temporal integration. The second normalizes, mapping a wide variety of occurrences of knowledge, and indicator elements such as (IF), (BECAUSE), (THEREFORE), (THEN), (OR), etc., into a single element such as (BECAUSEOF ...) or (COND...). The third does the actual grouping. During the course of these three steps all the indicators are assimilated one way or another. Some indicate the relationship of input or output. Others (e.g., (), the empty element) indicate a break in the verbal stream, so that a single operator group cannot span this. Thus, some groups are formed only with knowledge elements, as in the third protogroup in Figure 13.

One effect of the first step of the grouping process is to combine information that existed in adjacent topic segments. This can be seen in Figure 13, at the left, where the occurrence of (DIGIT 2) is combined with the prior occurrence of (EQ G 1) to give (MEQ G 1 2), i.e., "G must equal 1 or 2." Other forms of recombination also occur, e.g., (NEG) and (EQ G *D) in the same segment become (NEQ G *D), i.e., "G is not equal to some unknown digit."

The source of the rules used by the Semantic Processor is the limited task environment in which the subject is working. G cannot be 1 and 2, so it must be 1 or 2. Digits tend to be mentioned only in connection with letters; more strongly, if a letter is in the immediate neighborhood, the probability that it is associated with the digit is quite high. The source of the final grouping (step 3), is the basic assumption that everything can be described in terms of operators and their inputs and outputs and that mention of inputs and outputs occurs in the immediate neighborhood of the operator.

The Group Processor

After grouping has taken place, the next stage, the Group Processor, attempts to obtain a complete picture of what the subject knows at each moment and what operators he applies. This stage consists of two main parts, the first (the Determine Unknowns Mechanism) attempting to fill in unknowns in existing operators and knowledge elements, the second (the Origin Mechanism) attempting to infer operators and knowledge that were not verbalized by the subject during the experimental session.

The first part is the analog of anaphoric reference in the system. Many of the elements created by the Linguistic Processor have variables in them (denoted *L, *D, *C, etc.). Examples occurred in Figure 10 (c and d). During this step an attempt is made to match incomplete elements (elements with variables) against the possibilities defined by the current context. One possibility is that an element identical to the candidate already exists in the knowledge state. Then, the value is simply filled in, as shown in Figure 14 (a). The knowledge state is defined at this level by accessing the PBG, which is kept updated.

A second possibility is that the candidate is concerned with the processing of a column. The various columns are considered and an estimate made of how well the candidate fits the column; if the fit is close enough then the value of the variable is determined by matching to the appropriate element generated from the column. Figure 14 (b) illustrates this process for operator element (PLUS A *L) and knowledge element (EQ T *D). The unknown for the operator element is found by direct comparison with the letters in the columns. However, the unknown for the knowledge element is found by processing the columns containing T (in this case only column 1) in a one-step attempt to find its value. No attempt is made to determine the values of unknowns directly in terms of prior linguistic representations. It is more profitable to work in terms of the good semantic representation at hand, the PBG.

The second part of the Group Processor, the Origin Mechanism, attempts to posit operators and knowledge that did not occur in the linguistic representations. The basic generator of these inferences is the principle that each operator has inputs and outputs and that all knowledge was produced earlier as the output of some operator. Also involved is a continuity principle that knowledge once produced is available in the knowledge state thereafter.* These two principles permit us to infer, for any knowledge, the existence of an operator that produced it, and for any operator the existence of knowledge for inputs and outputs that are compatible with it.*

Table 2 gives a list of knowledge elements and the operators which can generate them. To infer an operator given its output we test each operator (defined as a possible candidate by the table) to see if it could generate the output when subject to the constraints of the current problem situation. Of the operators which pass this test, the one whose inferred inputs are most consistent with the current knowledge state is chosen as the most likely generator of the output. The process now continues recursively, as operators for generating the inferred inputs are themselves inferred.

Figure 15 shows how this works. At the top of the figure we have the knowledge state that is assumed, and below it the operator group under consideration. The top of the tree is the knowledge element whose origin is to be determined; it is part of the operator group. The tree itself is composed of operator groups which overlap such that the output of one operator may also be one of the inputs to another operator. For example, at the first level the leftmost group consists of operator (PC 6), inputs (EQ C6 0) (EQ D 5) (EQ G 2), and output (EQ R 7) (i.e., operator PC on column 6 with D=5, G=2 and carry=0 produced R=7). Each group at the first level represents a different hypothesis that could have produced (EQ R 7). At the lower levels the groups represent hypothesis that could have produced the inputs to the higher level groups. The tree is generated in a breadth-first fashion, and at each level the decision about which path to take is based on a measure of the agreement between the inputs for each path and the current context. In Figure 15 the encircled branches show the path chosen to represent the origin of (EQ R 7). These branches indicate that a PC on column 1 with C1=0 and D=5 produced C2=1, an AV produced L=3, and a PC on column 2 with C2=1 and L=3 produced R=7.

---

* This continuity principle can be modified to take into account separate memories, so that the principle applies only to Short-Term Memory, subject to a limited capacity, and that parts of the knowledge state stored in other memories (Long-Term Memory or External Memory) must be retrieved by recall operators. But these complications are not considered here.

* The subject could possibly make an error in applying an operator. However, the concept of problem space implies that it is used only if the operators can be applied with reasonable reliability. Thus, in general, errors in operator function are rare events and cannot be predicted.

The PBG Generator

The final stage of PAS-I generates the PBG. It is evoked whenever an operator group has been produced by the Group Processor. Due to the operation of the latter, a chain of groups, each with completed input and output elements and operators, may be produced at one time.

The PBG Generator works as follows. It takes a single operator group, consisting of one operator and its associated input and output elements and incorporates it into the existing PBG. In the simplest case the group is merely tacked on to the growing end of the PBG. However, if there exists a direct inconsistency between one of the output elements of the group and any currently active output element in the PBG, a restructuring of the PBG must occur. A knowledge element (and its node) is considered currently active if it belongs to a node lying along the lower (growing) edge of the PBG tree. Thus the conjunction of all currently active output elements constitutes the current knowledge state. PBG growth consists simply of adding a new node to the last currently active node in the tree. PBG restructuring consists of abandoning nodes (or groups of nodes) by redefining the location of the last currently active node. Thus restructuring is equivalent to returning to a prior point in the problem space, i.e., a prior knowledge state.

The rationale for restructuring is the following. As the subject traverses the problem space he may discover contradictions in his solution, or perceive that certain information is irrelevant. He will then abandon all information which initiated the contradiction or was found irrelevant, thus returning to some previous knowledge state. This abandonment or backing-up procedure is what makes the PBG tree structured.

An example of PBG growth is given in Figure 16.* In this artificial example** the input under consideration is the set of operator groups shown at the top of the figure. The first five groups are, in fact, the ones which the example of Figure 15 produces. Figure 16 shows the PBG at two stages: after the growth of 7 and 9 groups. The output of group 8 conflicts with that of node 5, leading to the abandonment of nodes 4, 5, 6 and 7. Note that value assignments (in this case node 4) which lead to conflicts are eliminated as well as the conflicting information itself.

We have traced through the operation of PAS-I, primarily by example. It generates a description of the behavior of the subject, given the input linguistic representation and also the structural models of the linguistic rules and the problem space. The space of possible descriptions is sufficiently rich that a genuine inferential procedure is required to find one adequate description. We have not, at this stage of development, attended to whether there exist alternative descriptions within the space and, if so, how to choose a preferred one.

V. Description of Behavior: Obtaining Topic Segments

PAS-I takes the topic segment as input, though the lexical representation (the sequence of words) would appear more natural. The reason for not extending the analysis back another stage is that the appropriate lexical representation is missing.

The fundamental basis for topic segmentation is twofold: the nature of English, where elementary expressions usually involve a single topic; and (more fundamentally) the serial nature of human information processing at this level of cognitive behavior. The subject attends to one thing at a time; consequently he will have a single topic to comment upon if he follows the instructions of Figure 1. (Some confusion between adjacent topics may occur, but this does not alter the basic situation.)

The segmentation can be made on the basis of three sources of knowledge: task structure, syntactic structure, and prosodic structure (i.e., pauses, breaks, stress, intonation). These provide substantial redundancy, so the problem does not appear difficult. From the task, there should be reference to no more than one variable type (i.e., letter or carry) and one value type (digit, even-odd, etc.). A topic can contain one of each, of course, since it often expresses a relation between a variable and a value (e.g., D is 5). Certain things are lost by this, e.g., disjunctive notions, such as "R could be 7 or 9," but in PAS-I later mechanisms compensate for this. From

* The PBG in Figure 16 is essentially a dual representation of the one in Figure 4. Figure 4 has nodes for knowledge states and branches for operators; Figure 16 has nodes for operator groups and branches for the resulting states of knowledge. The two representations carry the same information. Though both figures deal ostensibly with the same segment of behavior (Figure 2), they are both artificial examples for purposes of illustration.

** The companion paper (25) contains examples from actual protocols.

syntax, a topic should have a single verb and not extend over sentence boundaries. From the prosodic information, boundaries between topics are generally indicated by breaks, pauses, and downward intonations. Using just these three principles, without refinements, the entire protocol of S3 could probably be segmented into topics 75% correctly.

Much of this information is contained already in the punctuation, as it comes from the human transcriptionist. Thus, given the punctuation, topic segmentation appears almost too easy. On the other hand, without punctuation we have the lexical representation as a sequence of words, and the task of topic segmentation appears to become quite difficult. In this form the task is artificially hard, since the transcriptionist had available not only the sequence of words, but also prosodic information as well as meaning. Thus, it is not reasonable to attempt the task mechanically until a lexical representation is available that incorporates prosodic information as well as lexical items.*

VI. Description of Behavior: Trace of the Production System

PAS-I stops with the PBG, not because of the difficulty of proceeding further, but simply as the current state of development. The next behavior description task is to produce the trace of a production system (recall Figure 5) given the PBG, the problem space, and a production system.

This task seems easier than the one done by PAS-I. The production system, being a complete program can be run by a suitable interpreter (as illustrated in Figure 6) to produce a trace of the changing knowledge state. The task seems to be simply one of simulation, but in actuality it is more complex.

First, the trace must be identified with the behavior given by the PBG. Both the production system and the PBG (i.e., the given data) are imperfect. Consequently, the task of creating the trace requires matching it at every stage to the PBG and dealing with exceptions. Further, the trace may contain several steps for each one in the PBG. For example, the production system may predict the occurrence of operators that simply were not picked up in the PBG from the verbal behavior. Thus, a failure to match at a given step is not conclusive, since convergence may occur if additional steps are taken.

Second, the production system may embody a more detailed model of the information processing than is used for the problem space. This means that the trace could contain operators that never occur in the PBG. For instance, in the manual analysis of S3 the problem space was given in terms of four operators (PC, AV, GN and TD, as shown in Figure 3). The production system added to this additional operators whose function was attention direction or recall (e.g., FC, find column and FA, find antecedent expression). These operations are often not explicit in the verbal behavior and only become evident when a complete model of the process is attempted.

Third, the production system may be incompletely specified. This often arises because the operators themselves are incompletely specified. For example, the problem space defines PC by giving only the types of input information it can use and produce (knowledge elements associated with a specific column). It does not define the fine structure of the operator. A production system may add to this definition a program that works whenever actual digits are available (e.g., producing T=0 in column 1, D+D=T, if D=5 is given). But PC may remain undefined in other cases (e.g., in column 2, L+L=R, where carry=1, but nothing is known about L).

A scheme to handle these three problems has the following components:

> An interpreter of production systems that generates the next line of trace. This line may have symbolic indicators in it for outputs that could not be computed due to lack of specificity.
>
> A match routine that compares a line of trace with a knowledge state of the PBG:
>
> > If the two are identical where definite data is given, and the PBG data passes all tests associated with any incomplete operators in the trace then advance to the next node of the PBG and let the interpreter advance to the next trace line.

---

* Another artificial problem is disambiguating sentences such as "Suppose I make this a 6" versus "Suppose I make this A 6," or in general distinguishing between "a" and "A", "be" and "B", "Gee" and "G", "are" and "R", etc. In these cases the auditory representations contain additional clues to recognition that are lost if one simply considers the sequence of lexical items. Therefore, we do not attempt such disambiguation yet.

If the PBG data is not identical to the trace, and yet is not inconsistent with it, advance the trace only.

If the PBG data and the trace are inconsistent, fail.

A back-up mechanism that permits the decisions of the match routine to be tentative, so that alternative matchings of trace to data can be tried.

Below are examples of identity, consistency, and inconsistency, assuming that D=5 and C2=1 have already been established as elements in the trace and PBG.

Trace	PBG	Comparison
(PC 1)(EQ T 0)	(EQ T 0)	identical
(PC 1)(EQ T 0)	(EQ T 6)	inconsistent
(PC 2)	(ODD R)	consistent
(PC 2)	(EQ G 1)	inconsistent

Note that (ODD R) passes the tests associated with the incomplete operator PC, but (EQ G 1) does not.

This scheme does not contain any general mechanism for putting a simulation back on the track after error. But it is responsive to fitting the partial results of the production to the existing data in the PBG. As a side effect it produces a sequence of stipulated outputs of the incomplete operators. The usefulness of this sequence will be discussed in the next section.

Implementing the above scheme is not a task of the magnitude of that accomplished by PAS-I. It would produce, however, a sophisticated simulator, capable of working jointly with an imcompletely specified production system and with the PBG data that the system has to match.

VII. Induction of Rules

The description of behavior faces certain issues of inductive inference: what a given lexical sequence means and what knowledge a person possesses at a given moment. Inducing the various rule structures from the behavior faces different issues. Since we do not yet have operational programs for these inductive tasks, we are limited to framing specific problems. We will discuss briefly the induction of operators, the induction of productions and the induction of the problem space. We will not discuss the induction of linguistic rules.

Induction of operators

The problem space defines the general characteristics of an operator -- essentially its range and domain -- but does not define the action input/output relation. For example, from the problem space of Figure 3 we know that PC processes columns, using information about the letters and carries associated with a column and producing new information about associated letters and carries. But we have not defined the output it will produce from a specific set of inputs.

Given the successful formation of a PBG, a series of exemplars is obtained of the action of an operator. A portion of such data for the session of Figure 2 is shown in Table 3 (the full table has 76 entries). The task is then the following. Find a process that will work for all inputs of the form shown and will produce the outputs shown when given the corresponding inputs. The data need not be consistent. Thus, it is permissible to designate exceptions or to partition the input-output table as deriving from several distinct processes.

As in many induction tasks, trivial solutions are possible. Since the input-output table is finite, the table itself could be taken as memorized. This is equivalent to saying the subject does not calculate the result, he simply knows it. For example, in item 1 of Table 3 (D=5 and carry = 0 in column 1) he simply knows that 5+5=0 with 1 to carry. Likewise, in item 2 (carry=1 and L+L=R in column 2) he simply knows that R is odd.

This solution is unsatisfactory, since we believe the subject must process information to arrive at certain results. Item 1, which appears to involve just the addition table, might plausibly be memorized; item 2 would seem to require processing.

Thus, additional conditions must be placed on the induction task. One possibility is to consider the operator itself as a miniature production system with its own special set of operators. Then memorization can be equated with having a production (i.e., a condition-action rule) that yields a result directly in terms of the inputs. For example, letting (operand d) indicate that the number d is labeled an operand and, similarly, (sum d) that d is labeled a sum, i.e., a result, then the following productions would be admitted:

```
(operand 1) (operand 1) --> (sum 2)
(operand 1) (operand 2) --> (sum 3)
   ...          ...           ...
(operand 9) (operand 9) --> (carry 1) (sum 8).
```

These productions represent the basic addition table. However, no production like the following would be admitted:

(operand 1)(operand X)(operand X) --> (sum odd).

This task of induction is non-trivial (1,2,3). For instance, in prior analyses of S3 (by hand) two different programs for the column processing operator have been induced (13; 17, Ch. 6), neither of which is entirely adequate to represent the data of Table 3. Yet the task has a closed character that makes it amenable to the inductive techniques used elsewhere in artificial intelligence. Furthermore, if one considers the corresponding tables, not for PC, but (say) for the operator that generates all values of a variable defined by a given set of relations, (e.g., generate R for R odd and R>5), the task appears easier. For instance, one table for the generate operator (13) showed that the values generated were always correct (i.e., satisfied the given relations) and almost always went from low values to high. These two specifications essentially defined the process.

<u>Induction of the production system</u>

The information given is the PBG, the set of nodes giving the knowledge state at each point in time and the operator that advanced (or modified) that knowledge state. The desired result is an ordered set of productions which, when applied at each node, lead to the evocation of the operator that in fact occurs at that node.

The basic space of productions is comprised of those that can be formed in some production language. Its conditions are in terms of knowledge elements; its actions are in terms of operators with inputs specified by some operand identification procedure associated with matching the condition. Although we have not designed a production language for our automatic system, a formal version of this type of language can be found in (17, Ch. 2).

As before, we could make a large input-output table, with one entry for each node of the PBG. The input would be the total knowledge state at the node; the output would be the operator at the node (not the operator's output). Then a trivial solution is the production system that has a separate production for each node, namely, the one with condition equal to the knowledge state and action equal to the operator.

This, however, is a useful trivial solution. It permits posing the problem of induction of the production system as the problem of constructing a set of common subroutines. That is, the problem is how to rewrite the set of N productions (N, the total number of nodes) as a set of K (much less than N) parameterized productions. A natural way to proceed is by incrementally attempting to reduce the number of productions. Two productions with the same actions are compared on their conditions (i.e., the knowledge states), looking for the common elements. Additional clues exist, e.g., that an evoked production probably uses the information that was just added to the knowledge state. The problem of the induction of a production system has already been investigated relative to machine learning of heuristic (23, 24) and some of these techniques appear applicable.

An alternative approach (the one that scientists appear to use) is to hypothesize a general form for a production and then see how many situations it fits. This raises an important point about induction problems: the problem is never posed in an unstructured way. There is always a space of possibilities that is evoked on the basis of past experience and knowledge (and whose selection constitutes in some sense the real inductive leap). Thus, after only a few analyses (such as the manual ones already accomplished), much is known about the general character of production systems in cryptarithmetic. For instance, almost every subject has a production that is concerned with making use of new information, i.e., a producion of the form:

(EQ *letter digit*) --> (FC *letter*),(PC *column*)

like P1 of Figure 4. Similarly, all subjects have a production for backing down the tree, going from the contradiction of one fact to the contradiction of the antecedent fact. Knowing such productions exist reduces the task of induction considerably, since specific searches can be made for nodes where these productions are evoked. Currently, such productions exist as particularized variants for each experiment studied, but generalized forms do not seem difficult to obtain. Even without a generalized form, strong clues exist concerning which nodes would be candidates for the evocation of such productions, hence which subset of nodes should be collected for attempting, as a subtask, the induction of (say) a "use new information" production.

The induction of the production system takes on a form distinct from the induction of operators (which is the more general form of inducing a function from its input-output table). The reason is that productions were chosen to express models of human subjects because of their factorability into a series of independent pieces. Thus, the form of the process (as a set of productions) is already fixed and does not have to be induced from the data.

Induction of the problem space

We assume that the subject is operating in some problem space. The question is to determine its nature: what kinds of knowledge can the subject have and what sorts of operators does he apply to obtain it.

The major issue (as with all induction problems) is what is known of the space of all problem spaces. We know, by definition, that they consist of a set of knowledge and operator elements. Further, we know these both relate to the task of cryptarithmetic, and we have good linguistic grounds for positing how it will be talked about. For example, the subject will refer to "N", rather than to "the-stick-like character with two verticals and one diagonal." If such linguistic assumptions are violated, we have a more difficult task of induction.

It appears to be the case in cryptarithmetic that examples of operator and knowledge elements occur in relatively isolated and simple linguistic contexts. Thus evidence can be gleaned for the induction where there is little language complexity or simultaneous occurrence of conceptual elements to complicate matters. Table 4 shows some of the topic segments from the protocol of S3 that appear suitable for this task.

This suggests an inductive program built around an elementary grammar and a dictionary composed of verbs, relation terms, and task terms (i.e., letters, names, words, numbers, positions, etc.). Working with open language requires a large dictionary with definitions relevant to the task, in this case cryptarithmetic. Then we can expect such a program to identify from a subject's protocol the collection of knowledge and operator elements he is using to define his problem space.

Creating a list of problem space elements is a useful first step. For the problem space affects the entire protocol analysis sketched in Figure 9. It directly influences the operation and organization of the Linguistic Processor, the Semantic Processor, and the Group Processor. If a quite new problem space were obtained by the above procedure, how would the analysis of Figure 9 be carried out? Operational success in inducing the problem space lies not just in recognizing the elements, but in knowing how to use them -- i.e., how to integrate them into the analysis. This part of the question is clearly premature, for we have only begun to develop operational notions of how the problem space effects our analysis, and are in no position to rise above this to programs that create protocol analysis schemes.

VIII. Conclusion

We have attempted to lay out the task of protocol analysis as a field for work in artificial intelligence. Our base is rather narrow: protocol analysis in cryptarithmetic according to a particular style (17). Our reasons for this narrow base were set out in some methodological preliminaries. But even on this narrow base a wide range of intellectual scientific activities emerges: description of behavior, recognition of speech, induction of rules and structure, fitting of parametric models, generalization of models, prediction of behavior, and assessment of validity. We attempted to give substance to these tasks, starting with the description of behavior, for which we have a running system, PAS-I. We followed this with discussions of the tasks that, on the basis of current work, seem somewhat understood: extension of the behavioral description down toward the lexical level; extension up toward the production system trace; and induction of rules. The other tasks appear currently to be more remote.

The task of protocol analysis is a real one in experimental psychology, existing independently of any interest in it as a task in artificial intelligence. Unlike many tasks that currently hold central fascination in artificial intelligence, protocol analysis exhibits a lack of formality and an inherently inductive character that seems to characterize much other scientific (and real world) activity. Even Dendral (5), which is the closest attempt so far to deal with a complex scientific intellectual activity in artificial intelligence, rests heavily on the formality and tidiness of its empirical domain (chemical structures and numerical measures of their spectra). Protocol analysis is nowhere near so tidy. However, it too rests on certain simplicities -- e.g., the simplicity of the cryptarithmetic task itself. Thus, it is simply one step further along the road toward the full spectrum of scientific activity.

PAS-I currently does but a single task, however strongly one might feel that this task is intellectually significant. One purpose in emphasizing the spectrum of tasks encompassed by protocol analysis (recall Figure 8) is to note that serious, professional, long-term intellectual activity is not a single monolithic endeavor. Rather, it is a collection of interrelated tasks, tied together by common representations, common sources of knowledge and common memory of methods, heuristics, solutions, and difficulties. Soon we must come to grips with such intellectual conglomerates.

Acknowledgments

Discussions with H. A. Simon have been of much benefit throughout this work. This research has been supported primarily by Research Grant MH-07722 from the National Institute of Mental Health and in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44610-70-C-0107) and is monitored by the Air Force Office of Scientific Research.

References

1. Amarel, S. An approach to automatic theory formation. In von Foerster, H. (ed.), Illinois Symposium on Principles of Self-Organization, University of Illinois, 1962a.

2. Amarel, S. On the automatic formation of a computer program which represents a theory. In Yovits, M. C., Jacobi, G., and Goldstein, G., (eds.) Self-Organizing Systems, Spartan Books, Washington, D. C., 1962b, pp. 107-175.

3. Amarel, S. Representations and modeling in problems of program formation. In Meltzer, B., and Michie, D. (eds.), Machine Intelligence 6, American Elsevier, 1971.

4. Bree, D. S. The understanding process, as seen in the geometry theorems and the Missionaries and Cannibals problem. Ph.D Thesis Psychology Dept., Carnegie-Mellon University, 1968.

5. Buchanan, B., Sutherland, G., and Feigenbaum, E. A. Heuristic Dendral: a program for generating explanatory hypotheses in organic chemistry. In Meltzer, B., and Michie, D. (eds.), Machine Intelligence 4, American Elsevier, 1969, pp. 209-254.

6. DeGroot, A. D. Thought and Choice in Chess. Basic Books Inc., New York, 1965.

7. DeGroot, A. D. Perception and memory versus thought: Some old ideas and recent findings. In Kleinmuntz, B. (ed.) Problem Solving: Research, Method, and Theory, Wiley and Sons, New York, 1966, pp. 19-50.

8. Evans, T. G. A program for the solution of a class of geometric-analogy intelligence test questions. In Minsky, M (ed.) Semantic Information Processing, MIT Press, Cambridge, Mass., 1968, pp. 271-353.

9. Feldman, J. Simulation of behavior in the binary choice situation. In Feigenbaum, E., and Feldman, J. (eds.), Computers and Thought, New York, McGraw-Hill, 1963, pp. 329-346.

10. Hunt, E. B., Marin, J., and Stone, P. J. Experiments in Induction. Academic Press, 1966.

11. Johnson, E. S. An information processing model of one kind of problem solving. Psychological Monographs, vol. 78, no. 4, whole no. 581, 1964.

12. Newell, A. On the analysis of human problem solving protocols. In Gardin, J. C. and Jaulin, B. (eds.), Calcul et Formalisation dans les Sciences de L'Homme, Centre National de la Recherche Scientifique, 1968, pp. 146-185.

13. Newell, A. Studies in problem solving: Subject 3 on the crypt-arithmetic task DONALD+GERALD=ROBERT. Computer Science Dept., Carnegie-Mellon University, 1967.

14. Newell, A., and Simon, H. A. Computer simulation of human thinking. Science, vol. 134, no. 3495, 1961, pp. 2011-2017.

15. Newell, A., and Simon, H. A. GPS, a program that simulates human thought. In Feigenbaum, E., and Feldman, J. (eds.), Computers and Thought, McGraw-Hill, 1963, pp. 279-293.

16. Newell, A., and Simon, H. A. An example of human chess playing in the light of chess playing programs. In Wiener, N., and Schade, J. P. (eds.), Progress in Bio-cybernetics, vol. 2, Elsevier Publishing Co., Amsterdam, 1965, pp. 19-75.

17. Newell, A., and Simon, H. A. Human Problem Solving. Prentic Hall, Englewood Cliffs, N. J., 1971 (in press).

18. Newell, A., Shaw, J. C., and Simon, H. A., Elements of a theory of human problem solving. Psychological Review, vol. 65, no. 3, May 1958, pp. 151-166.

19. Newsted, P. Toward a process-oriented theory of concept identification. Ph.D. Thesis, Psychology Dept., Carnegie-Mellon University, 1970.

20. Paige, J. J., and Simon, H. A. Cognitive processes in solving algebra word problems. in Kleinmuntz, B. (ed.) Problem Solving: Research, Method and Theory, New York, Wiley, 1966, pp. 51-119.

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

22. Simon, H. A., and Kotovsky, K. Human acquisition of concepts for sequential patterns. Psychological Review, vol. 70, no. 6, 1963, pp. 534-546.

23. Waterman, D. A. Machine learning of heuristics. Ph.D. Thesis, Computer Science Dept., Stanford University, 1968.

24. Waterman, D. A. Generalization learning techniques for automating the learning of heuristics. Artificial Intelligence, vol. 1, nos. 1 and 2, 1970, pp. 121-170.

25. Waterman, D. A., and Newell, A. Preliminary results with a system for automatic protocol analysis. Cognitive Psychology (to be submitted), 1971.

26. Winikoff, A. W. Eye movements as an aid to protocol analysis of problem-solving behavior. Ph.D. Thesis, Electrical Engineering Dept., Carnegie-Mellon Univ., 1968.

* * * * * *

```
  D O N A L D    D = 5
+ G E R A L D
 _____________
  R O B E R T
```

The expression at the side is a simple arithmetic sum in disguise. Each letter represents a digit, that is, 0, 1, 2, ..., 9. Each letter is a distinct digit. You are given that D represents the digit 5; thus, no other letter may be 5.

What digits should be assigned to the letters such that when the letters are replaced by their corresponding digits the above expression is a true arithmetic sum?

Please talk all the time while you work, saying whatever is on your mind at each moment, however fragmentary, trivial, apparently irrelevant, impolitic, or indiscreet. Whenever you fall silent for more than a moment the experimenter will ask you to "please talk."

FIGURE 1. Instructions for Cryptarithmetic Task

1. Each letter has one and only one numerical value --
2. Exp: One numerical value.
3. There are ten different letters
4. and each of them has one numerical value.
5. Therefore, I can, looking at the two D's --
6. each D is 5,
7. therefore, T is zero.
8. So I think I'll start by writing that problem here.
9. I'll write 5, 5 is zero.
10. Now, do I have any other T's?
11. No.
12. But I have another D.
13. That means I have a 5 over the other side.
14. Now I have 2 A's
15. and 2 L's
16. that are each --
17. somewhere --
18. and this R --
19. 3 R's --
20. 2 L's equal an R --
21. Of course I'm carrying a 1.
22. Which will mean that R has to be an odd number.
23. Because the 2 L's --
24. any two numbers added together has to be an even number
25. and 1 will be an odd number.

FIGURE 2. Initial Phrases of Transcription of S3 Problem Session

Knowledge Elements

l	:= A|B|D|E|G|L|N|O|R|T	Letters in the display
d	:= 0|1|2|3|4|5|6|7|8|9	Digits assignable to letters
c	:= C1|C2|C3|C4|C5|C6|C7	Carries into a column
col	:= 1|2|3|4|5|6|7	Columns (from right to left)
v	:= *l*|*c*	Variables: letters or carries
lset	:= *l*|*l lset*	Sets of letters
eq	:= EQ|AEQ	Equality relations
rel	:= EQ|AEQ|GR|SM|ODD|EVEN|PEQ	Relations
	(EQ *l d*)	*l* is inferred equal to *d*
	(AEQ *l d*)	*l* is assumed equal to *d*
	(GR *v d*)	*v* is greater than *d*
	(SM *v d*)	*v* is smaller than *d*
	(ODD *v*)	*v* is odd
	(EVEN *v*)	*v* is even
	(PEQ *v d*)	*v* is possibly equal to *d*

Operator Elements

(PC *col v*)	Process *col* for information about *v* (*v* is optional)
(AV *v*)	Assign a value to *v*
(GN *v*)	Generate the possible values of *v*
(TD *l d*)	Test if *d* is legal for *l*

FIGURE 3. Elements from the Problem Space for S3

P1: (*eq v d*) --> (FC *v*), (PC *col*)
P2: (GET *v*) --> (FC *v*), (PC *col v*)
P9: (GET *lset*) --> (FL *lset*), (GET *l*)
P11: (EQ *l d*) --> (TD *l d*)

Additional operators

(FC *v*)	Find a column containing variable *v*
(FL *lset*)	Find a letter in set *lset*

Additional knowledge elements

ltrs := (D T L R A E N B O G)	A set of letters
(GET *ltrs*)	The goal is to find the values of the letters in *ltrs*
(GET *v*)	The goal is to find the value of *v*

FIGURE 4. Simplified Productions from the Production System for S3 (Knowledge in the right side of a production, e.g., (GET *l*) is simply copied into the knowledge state.)

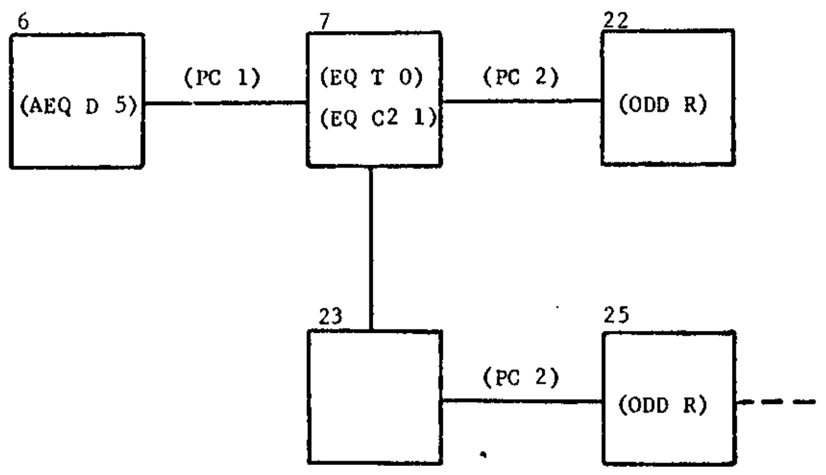


FIGURE 5. PBG for Initial Part of S3 Problem Session

PHRASE	PRD	OPR	RESULT	KNOWLEDGE STATE
				(AEQ D 5)(GET LTRS)
	P1	(FC D)		
5			1	(AEQ D 5)(GET LTRS)
6		(PC 1)		
7				(EQ T 0)(EQ C2 1)(AEQ D 5)(GET LTRS)
	P11	(TD T 0)		
			+	(EQ T 0)(EQ C2 1)(AEQ D 5)(GET LTRS)
10	P1	(FC T)		
11			-	(EQ T 0)(EQ C2 1)(AEQ D 5)(GET LTRS)
14	P9	(FL LTRS)		
18			R	(EQ T 0)(EQ C2 1)(AEQ D 5)(GET LTRS)
				(GET R)(EQ T 0)(EQ C2 1)(AEQ D 5)(GET LTRS)
	P2	(FC R)		
			2	(GET R)(EQ T 0)(EQ C2 1)(AEQ D 5)(GET LTRS)
20		(PC 2 R)		
22				(ODD R)(GET R)(EQ T 0)(EQ C2 1)(AEQ D 5)(GET LTRS)
	.	.		.
	.	.		.
	.	.		.

FIGURE 6. Trace of Production System for S3
(Order of evocation of productions cannot be derived from the partial set of productions shown in Figure 4.)

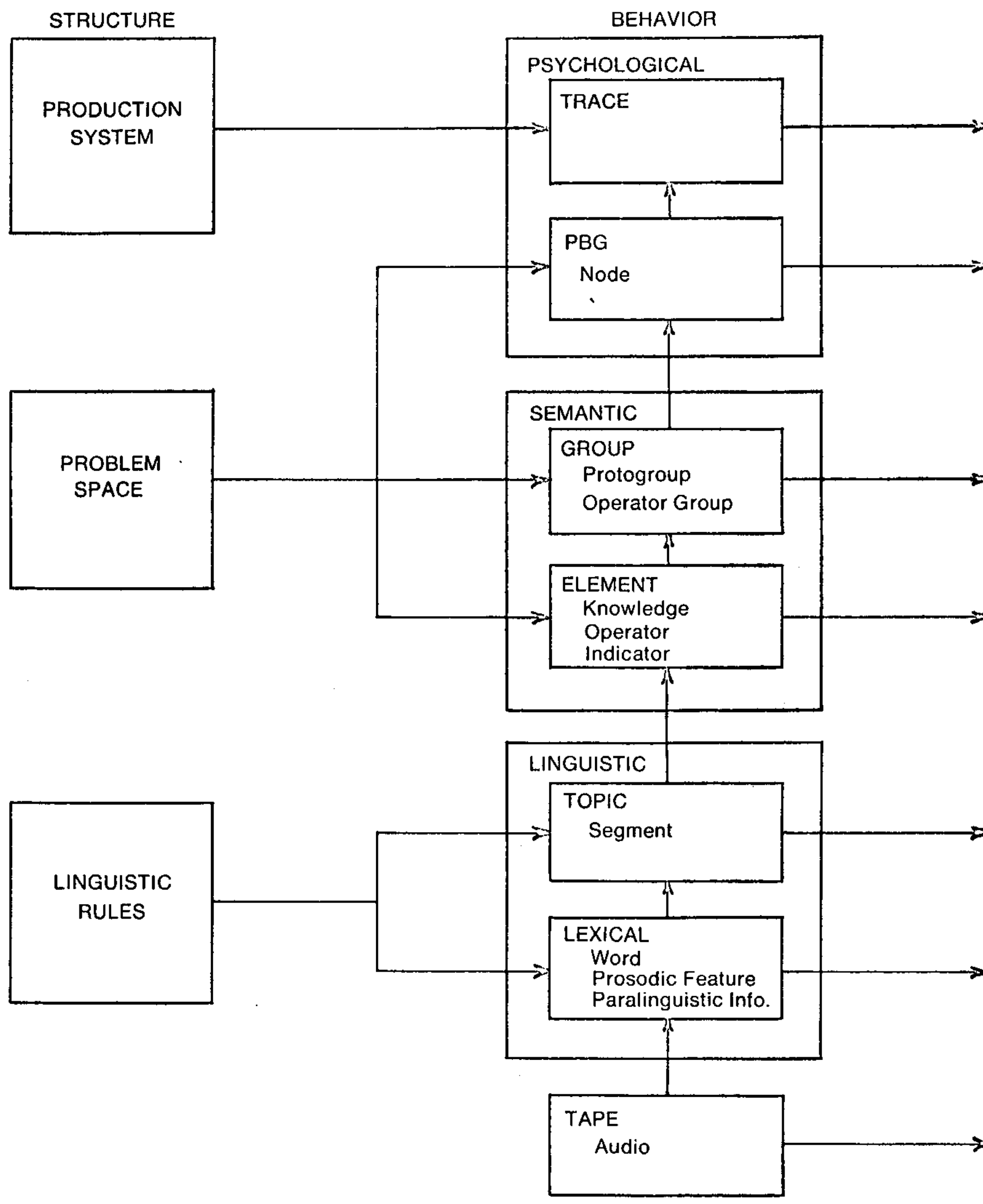


Figure 7. Representations for Protocol Analysis

<u>Description of behavior</u>: Find the representation of behavior at some level, given the representation of behavior at some lower level.

<u>Recognition of speech</u>: Find the lexical representation of behavior given the audio representation (special case of description).

<u>Induction of rules</u>: Find a static structure (linguistic rules, problem space, production system), given a representation of behavior.

<u>Fitting of models</u>: Find a static structure, given a representation of behavior and a class of structures described in a parametric or systematic way.

<u>Generalization of models</u>: Modify a static structure that is adequate for some set of behaviors to encompass a newly given behavior in some representation.

<u>Prediction of behavior</u>: Find the behavior in some representation, given some static structures along with the defining conditions for an experimental situation.

<u>Assessment of validity</u>: Find the validity, expressed in some representation, of a given static structure or behavior in some representation.

FIGURE 8. Varieties of Subtasks in Protocol Analysis

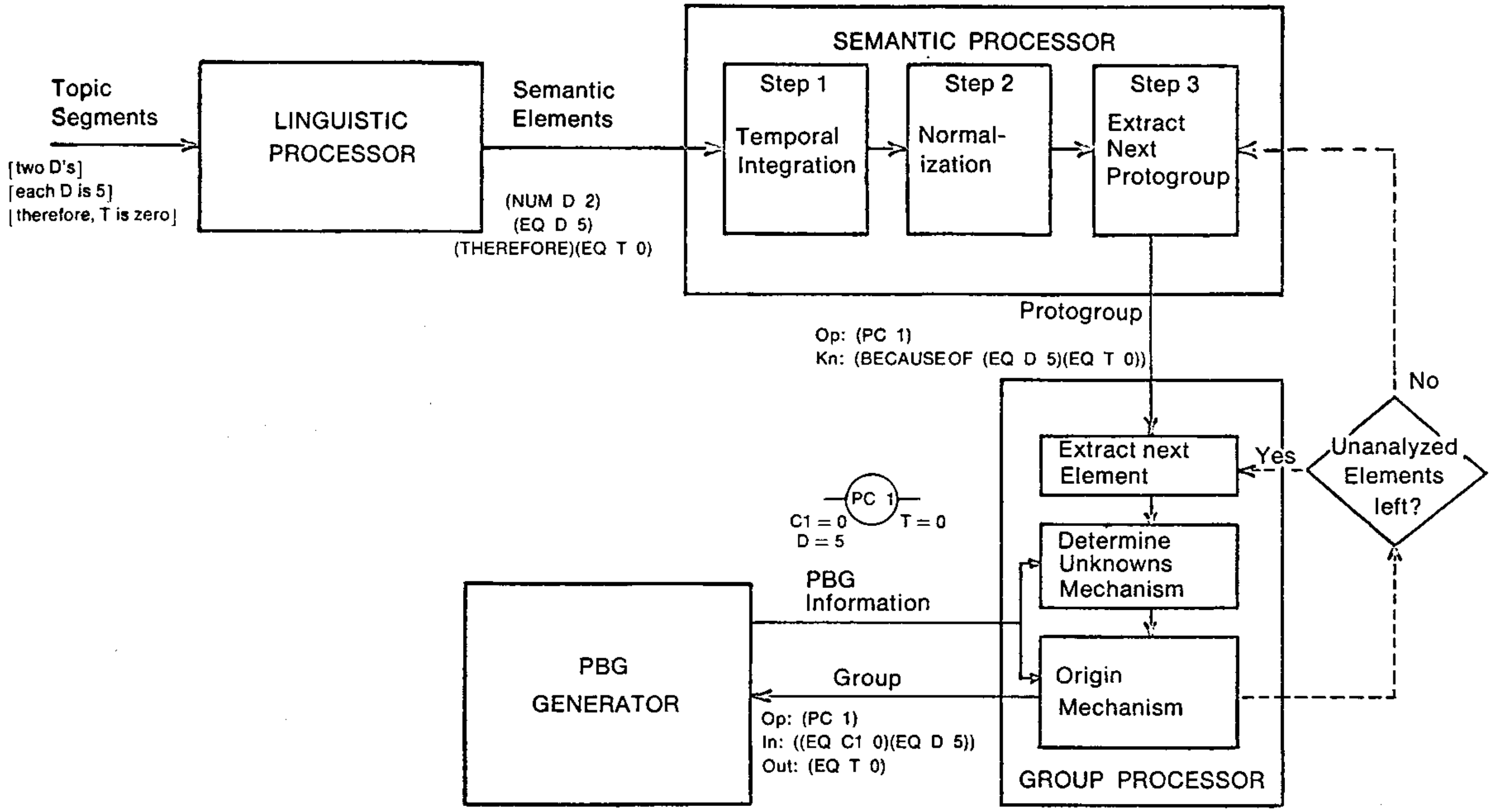


Figure 9. Flow Diagram of PAS-I

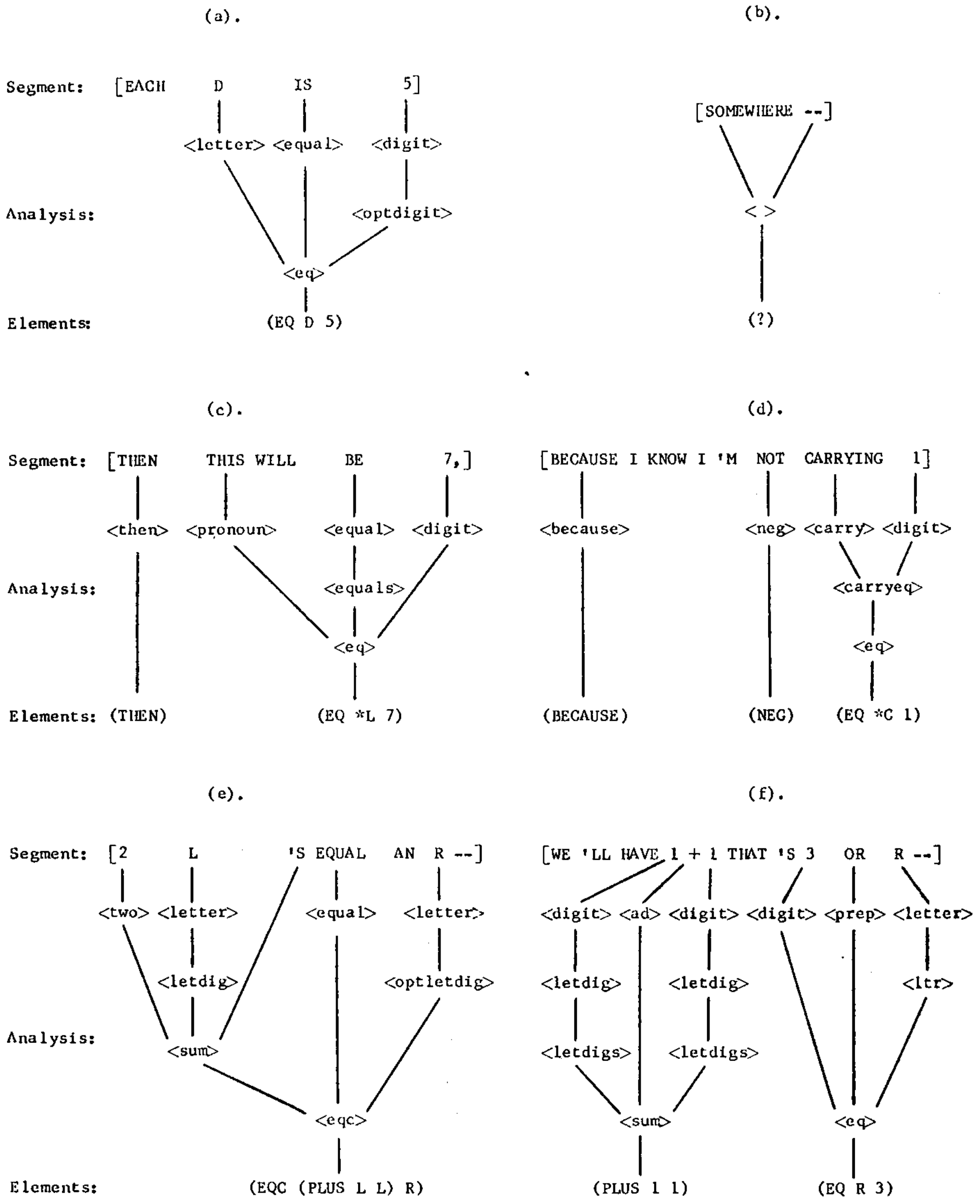


FIGURE 10. Examples of Linguistic Processor Operation

```
<eq> := <carryeq> | <letter> <equal> <optdigit> | <pronoun> <equals> <digit> | <digit><prep><ltr>
<sum> := <letdigs><ad> <letdigs> | <two><letdig>'S
<eqc> := <sum> <equal> <optletdig>
<carryeq> := <carry> <digit>
<ltr> := <pronoun> <letter> | <letter> <pronoun> | <letter> | <pronoun>
<optletdig> := <digit> | <letter> | < >
<optdigit> := <digit> | < >
<letdigs> := <letdig> | <pronoun>
<letdig> := <letter> | <digit>
<equals> := <equal> | 'S
<equal> := IS | BE | EQUAL
<letter> := D | L | R
<digit> := 1 | 3 | 5 | 7
<carry> := CARRYING
<because> := BECAUSE
<then> := THEN
<prep> := OR
<pronoun> := THIS
<neg> := NOT
<ad> := +
<two> := 2
```

FIGURE 11. A Subset of the Grammar Used by the Linguistic Processor

```
   5.  Therefore, I can, looking at the two D's --
 *16.  that are each --
 *17.  somewhere --
  24.  any two numbers added together has to be an even number
  25.  and 1 will be an odd number.
  38.  if I have to carry 1 from the E + 0.
 *50.  it's not possible that there could be another letter in front
       of this R is it?
  69.  and it's the L's that will have to be 3's,
 *72.  Now, it doesn't matter anywhere what the L's are equal to --
 *79.  that is, itself plus another number equal to itself.
 118.  Then again, that's assuming that N is less than 3,
*161.  in order to have the O = the O.
 202.  And also am using R as 9 instead of a 7
*230.  and it doesn't seem as though I'm going to be able to carry more
       than 1 in any case.
*282.  Of course, it all has to satisfy the fact that I have 10 letters
       for 10 numbers.
*286.  I'm only two numbers short, aren't I?
```

FIGURE 12. Types of Complex Utterances Analysed by the Linguistic Processor (The examples are taken from the protocol of S3; those marked with asterisks cannot be handled by the current system.)

```
Initial
Semantic Elements                                                                                    Protogroups

                                                                                                     1st protogroup

(NUM D 2)                        (NUM D 2)                                                           (NUM D 2)
(EQ D 5)                         (EQ D 5)                                                            (BECAUSEOF (EQ D 5)
(THEREFORE) (DIGIT 0)            (THEREFORE) (EQ *L 0)      (NUM D 2)                                           (EQ *L 0))
(LETTER R)                       (LETTER R)                 (BECAUSEOF (EQ D 5) (EQ *L 0)            (LETTER R)
(UNLESS)             STEP 1      (PLUS D G)       STEP 2    (LETTER R)                      STEP 3   2nd protogroup
(PLUS D G)           ====>       (MEQ G 1 2)      ====>     (PLUS D G)                      ====>
(EQ G 1)                         (BECAUSE) (EQ D 5)         (BECAUSEOF (EQ D 5) (MEQ G 1 2))         (PLUS D G)
(DIGIT 2)                        ( )                        ( )                                      (BECAUSEOF (EQ D 5)
(BECAUSE)(EQ D 5)                (IF) (EQ *L 7)             (COND ((EQ *L 7) (EQ D 5))(NEQ G *D))               (MEQ G 1 2))
(?)                              (AND) (EQ D 5)             (EQ C6 1)
(IF) (EQ *L 7)                   (THEN) (NEQ G *D)                                                   3rd protogroup
(AND) (EQ D 5)                   (EQ C6 1)
(THEN) (NEG) (EQ G *D)                                                                               (COND ((EQ *L 7)(EQ D 5))
(CIN 1 (PLUS D G))                                                                                         (NEQ G *D))
                                                                                                     (EQ C6 1)
```

FIGURE 13. Examples of Semantic Processor Operation

(a) Knowledge State: (EQ D 5)(GREATER R 7)(EQ C1 0)

Determine Unknowns

(EQ *L 5) $\Longrightarrow$ (EQ D 5)

(EQ *C 0) $\Longrightarrow$ (EQ C1 0)

(GREATER R *D) $\Longrightarrow$ (GREATER R 7)

(b) Knowledge State: (EQ D 5)(EQ C1 0)

```
            c6 c5 c4 c3 c2 c1
             D  O  N  A  L  D
Display:   + G  E  R  A  L  D
            ----------------
             R  O  B  E  R  T
```

Determine Unknowns

(PLUS A *L) $\Longrightarrow$ (PLUS A A)

(EQ T *D) $\Longrightarrow$ (EQ T 0)

FIGURE 14. Examples of Inferences by Determine Unknowns Mechanism

Knowledge State: (EQ D 5) (EQ C1 0) (EQ C7 0) (EQ G 4) (EQ C6 0)

Operator Group: Operator (PC 4) Elements (EQ R 7) (EQ L 3)

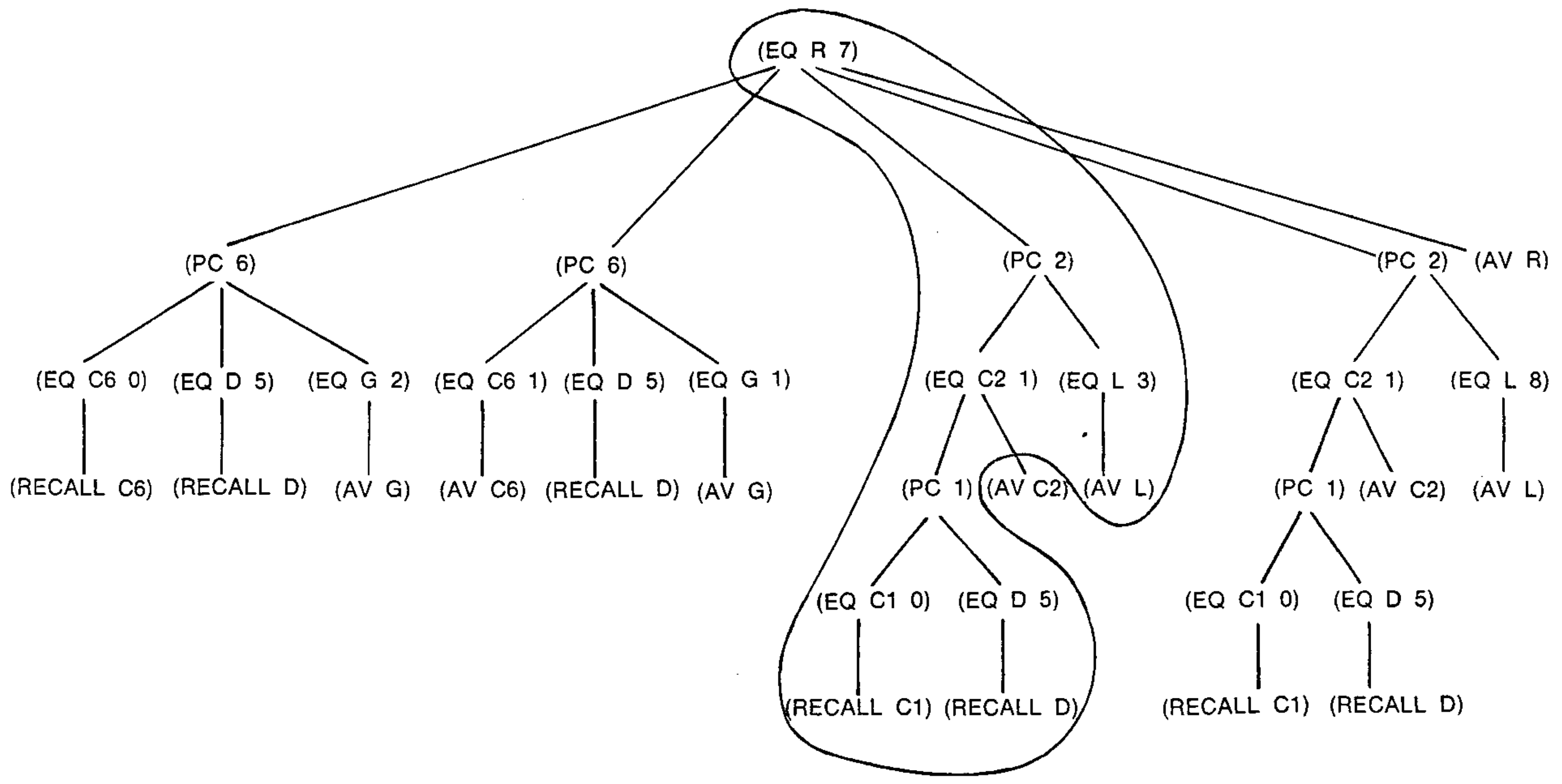
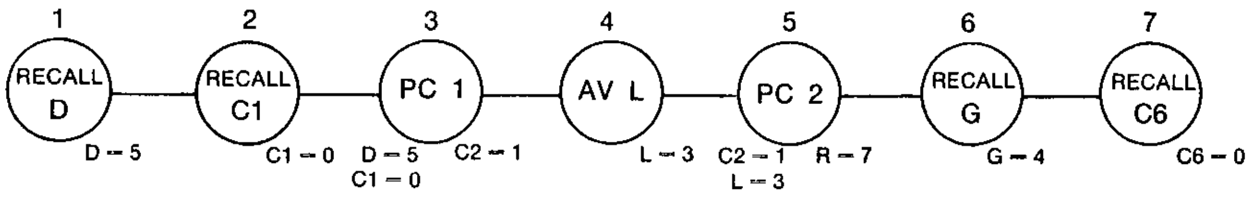


Figure 15. Operation of the Origin Mechanism

Initial or Given
Knowledge State: (EQ D 5) (EQ C1 0) (EQ C7 0) (EQ G 4) (EQ C6 0)

		Operator	*Inputs*	*Outputs*
Operator Groups:	(1).	(RECALL D)	( )	(EQ D 5)
	(2).	(RECALL C1)	( )	(EQ C1 0)
	(3).	(PC 1)	(EQ D 5) (EQ C1 0)	(EQ C2 1)
	(4).	(AV L)	( )	(EQ L 3)
	(5).	(PC 2)	(EQ C2 1) (EQ L3)	(EQ R 7)
	(6).	(RECALL G)	( )	(EQ G 4)
	(7).	(RECALL C6)	( )	(EQ C6 0)
	(8).	(PC 6)	(EQ D 5) (EQ C6 0) (EQ G 4)	(EQ R 9)
	(9).	(AV L)	( )	(EQ L 2)

Problem Behavior
Graph 1-7.



1-9.

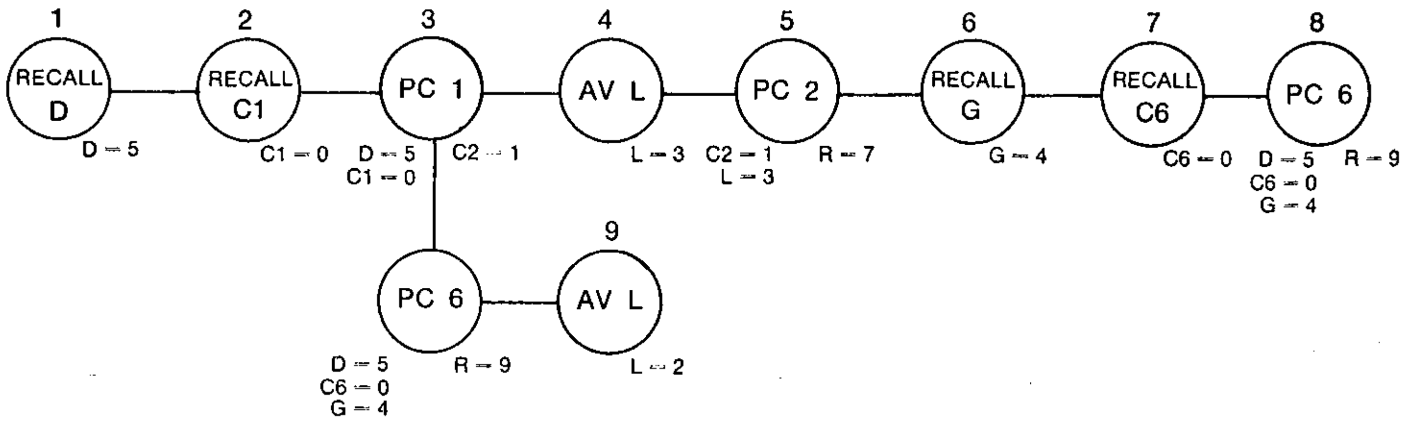


Figure 16. Example of PBG Generation

SEMANTIC ELEMENTS

KNOWLEDGE	MEANING	OPERATORS	MEANING	INDICATORS
(LETTER l)	An occurrence of the letter l	(FC v)	Find a column containing v	(OR)
(DIGIT d)	An occurrence of the digit d	(NUM l d)	the number of l's is d	(IF)
(PLS u)	u is added to something	(PLUS u_1 u_2)	u_1 is added to u_2	(AND)
(IN v d)	v is in column d	(EQC (PLUS u_1 u_2)u_3)	u_1 plus u_2 equals u_3	(YES)
(EVEN v)	v is even	(COUNT l)	Count the number of l's	(NEG)
(ODD v)	v is odd	(RECALL v)	Recall the value of v	(QUES)
(EQ v d)	v equals d	(PC d)*	Process column d	(THEN)
(PEQ v d)	One possible value for v is d	(GN l)*	Generate possible values for l	(BECAUSE)
(GREATER v d)	v is greater than d	(IG c)*	Ignore the carry c	(UNLESS)
(SMALLER v d)	v is smaller than d	(AV v)*	Assign some value to v	(ASSUME)
(CIN d col)	The carry into column col is d	(FA e)*	Find the antecedent of element e	(DIFFICULT)
(COUT d col)	The carry out of column col is d	(FN e)*	Find the negative of the antecedent of e	(THEREFORE)
(MEQ v d_1 d_2)*	v must equal either d_1 or d_2	(TD v d)*	Test if v can be equal to d	(CORRECTION)
(NEQ v d)*	v is not equal to d	(TE e)	Test if e can be true	(INSTEADOF)
(AEQ v d)*	v is assumed to have the value d			
(COND e_1 e_2)*	If e_1 is true then e_2 is true			

* These elements are generated by the Semantic Processor rather than the Linguistic Processor.

TABLE 1. Examples of Semantic Elements Used in PAS-I (l represents an arbitrary letter, d a digit, c a carry, v a letter or carry, u a letter, carry, or digit, e a knowledge element, and col an element such as (PLUS A A) which indicates a column.)

KNOWLEDGE ELEMENTS	OPERATORS
EQ	PC, GN, IG, FA, TD, TE, AV
PEQ	PC, GN, FA
MEQ	PC, GN, FA
NEQ	FN, TD, TE, PC
AEQ	FA, AV
EVEN	PC, FA, TD, TE
ODD	PC, FA, TD, TE
GREATER	PC, FA, TE
SMALLER	PC, FA, TE

TABLE 2. Knowledge Elements and Operators for Generating Them

	Inputs	Operator	Outputs
1.	(EQ C1 0)(EQ D 5)	(PC 1)	(EQ T 0)(EQ C2 1)
2.	(EQ C2 1)	(PC 2)	(ODD R)
3.	(EQ D 5)(ODD R)	(PC 6)	(EVEN G)
4.	(EQ C2 1)(EQ L 1)	(PC 2)	(EQ R 3)
5.	(EQ D 5)	(PC 6)	(GREATER R 5)
6.	( )	(PC 5)	(PEQ E 9)(EQ C6 1)
⋮	⋮	⋮	⋮

```
          c6 c5 c4 c3 c2 c1
           D  O  N  A  L  D
Task:    + G  E  R  A  L  D
          _________________
           R  O  B  E  R  T
```

TABLE 3. Input/Output examples for PC of S3

TOPIC SEGMENTS		SEMANTIC ELEMENTS
Knowledge		
6.	[EACH D IS 5 .]	EQ
12.	[BUT I HAVE ANOTHER D .]	IN
21.	[OF COURSE I 'M CARRYING UH 1 .]	EQ
22.	[WHICH WILL MEAN THAT R HAS TO BE AN ODD NUMBER .]	ODD
35.	[G HAS TO BE AN EVEN NUMBER .]	EVEN
96.	[R COULD BE 9 ALSO .]	PEQ
118.	[THEN AGAIN , THAT 'S ASSUMING THAT N IS LESS THAN 3 ,]	SMALLER
135.	[BUT A CAN N'T EQUAL 5 .]	NEQ
201.	[AND ALSO AM USING R AS 9 INSTEAD OF 7 .]	AEQ
213.	[AND R HAS TO BE GREATER THAN 5 .]	GREATER
Operators		
10.	[NOW , DO I HAVE ANY OTHER T 'S ?]	FC
15.	[AND 2 L 'S]	PC
130.	[A + A --]	PC
151.	[SUPPOSE O WERE 1]	AV
200. 201.	[OF COURSE NOW MY E CAN N'T BE A 9 ,] [SINCE I 'VE USED THE 9 FOR R .]	TD

TABLE 4. Topic Segments for Induction of Problem Space