PAS-II: AN INTERACTIVE TASK-FREE VERSION OF AN AUTOMATIC PROTOCOL ANALYSIS SYSTEM

D. A. Waterman and A. Newell

June, 1973

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

This paper will appear in the preprints for the third International Joint Conference on Artificial Intelligence (IJCAI-73). This research was supported in part by Research Grant MH-07732 from the National Institutes of Health and in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-70-C-0107) which is monitored by the Air Force Office of Scientific Research.

PAS-II: AN INTERACTIVE TASK-FREE VERSION OF AN AUTOMATIC PROTOCOL ANALYSIS SYSTEM

D. A. Waterman and A. Newell

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

Abstract

PAS-II, a computer program which represents a generalized version of an automatic protocol system (PAS-I) is described. PAS-II is a task-free, interactive, modular data analysis system for inferring the information processes used by a human from his verbal behavior while solving a problem. The output of the program is a Problem Behavior Graph: a description of the subject's changing knowledge state during problem solving. As an example of system operation the PAS-II analysis of a short cryptarithmetic protocol is presented.

1. Introduction

Automatic protocol analysis is a joint effort by man and machine to infer from the record of the time course of a subject's behavior, the underlying information processes. As developed (5), it usually refers to the verbalizations of a subject solving some problem under instructions to think out loud. Protocol analysis designates the full range of activities engaged in by the psychologist when working with protocols: description of the subject's behavior according to an hypothesized model, induction of new rules, derivation of consequences from a model in the context of specific data, and measurement of adequacy of a model. The initial focus of our work has been behavior description in terms of information processes, given an hypothesized general model (the so-called problem space in which the subject operates).

The PAS-I system (14, 15) was our first attempt at automatic protocol analysis. This is a fully automatic, non-interactive, specialized system designed to analyze cryptarithmetic protocols and produce as output a problem behavior graph (PBG)describing the subject's search through a posited problem space. The protocol analysis is represented as a sequence of processing stages that eventually transform the raw protocol into a problem behavior graph. At each stage rules are applied which effect a transformation of the data. The organization of PAS-I is shown in Figure 1.

PAS-I has successfully analyzed protocols from DONALD+GERALD=ROBERT and CROSS+ROADS=DANGER cryptarithmetic problems. The results obtained in the DONALD+GERALD=ROBERT task for two of the subjects have been discussed in detail (15) and demonstrate that this approach to automatic protocol analysis is both feasible and rewarding.

Encouraged by the success of PAS-I we have designed and built an improved version called PAS-II. PAS-II was designed with two major goals in mind: to make it interactive and task free. By interactive we mean that the user is permitted to take an active part in the analysis: he can provide answers to subproblems the system is unable to solve, correct processing errors, and even maintain control over the processing sequence. Clearly, real-time interaction of this sort makes the system a more powerful tool for protocol analysis. By task free we mean that the system is independent of any particular problem domain. To make PAS-II task free we partitioned the system into two parts: the problem dependent part consisting of the processing rules or heuristics used at each stage of the analysis, and the problem independent part consisting of the general control structure and command language. Thus, to apply the system to a protocol in a new problem area the user must first supply the system with processing rules for that domain.* The design of PAS-II also included four subgoals: to make the system transparent, modifiable, extendable, and open (see Figure 2).

Two important implementation issues were not addressed in the design of PAS-II. 1). Improve system performance in cryptarithmetic. This includes expanding the deductive and inductive inference capabilities, and "fine tuning" the system by optimizing the processing heuristics to produce the best possible analysis within the given framework. 2). Extend the scope of the analysis. For example, extend the system back to handle the speech recognition and segmentation problems inherent in producing a transcription from the audio tape. Or extend the system to handle the problem of inducing the problem space from the protocol or inducing a production system model from the problem behavior graph.

It was decided to make PAS-II interactive and task free, postponing the problems of increasing power in a particular task or broadening the scope of the analysis. This decision was influenced by the desire to provide a working tool for protocol analysis that could be used by participants at a workshop on New Techniques in Cognitive Research held at CMU in the summer of 1972 (7). The PAS-II system is currently running in LISP at CMU on a PDP-10 and is available to the CMU (and the ARPA Network) community.

This paper is organized as follows. The task of protocol analysis is discussed in Section 2. This is followed in Section 3 by a brief description of the structure of the program and in Section 4 by an example of its use in analyzing a cryptarithmetic protocol. Section 5 concludes with a discussion of the general executive structure of the system and its implication for AI data analysis programs.

2. Task of Protocol Analysis

Protocol analysis is a complex data processing task requiring both deductive and inductive inference capabilities. Our current approach to protocol analysis is based on a particular theory of human problem solving. For a description of this theory and an introduction to the task of protocol analysis see Newell and Simon (5).

---

* Ultimately, a library containing processing rules for a number of different problem domains will be available to the user.

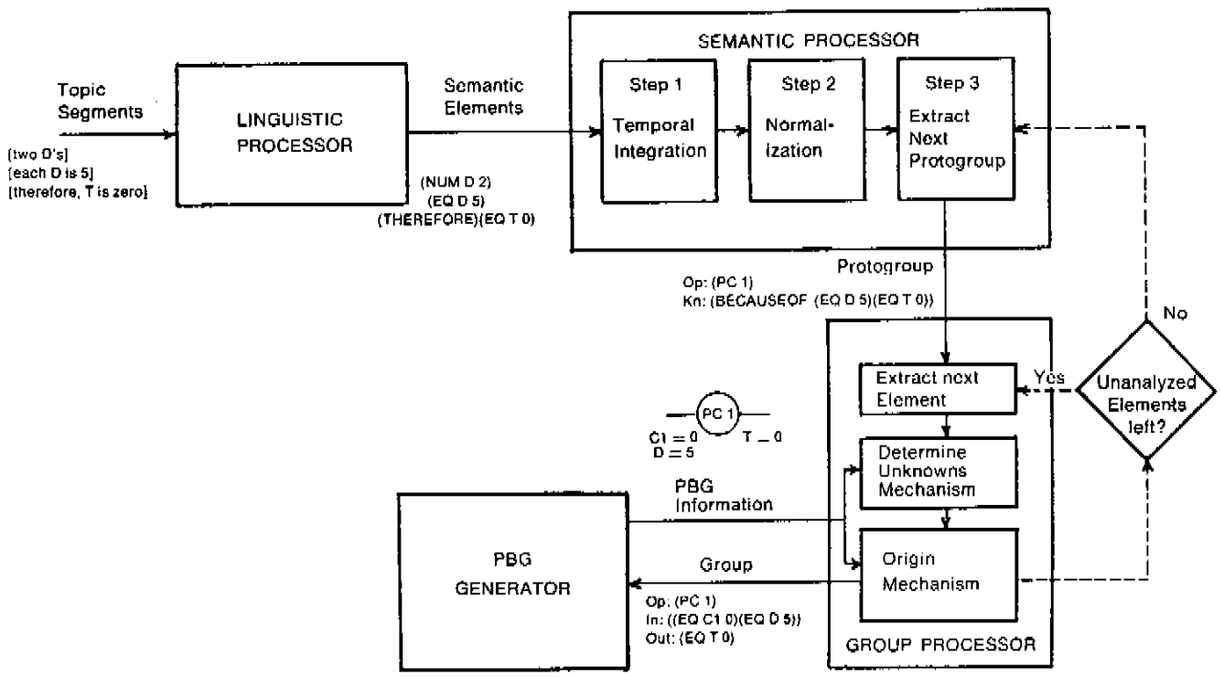

Figure 1. Flow Diagram of PAS-I.

GOALS

Interactive:		User and system exchange information during processing.
Task Free	:	System is independent of any particular problem domain.

SUBGOALS

Transparent:		System is easy to use and understand by virtue of a clean organization and the ability to explain itself.
Modifiable	:	Basic changes in the data processing procedure can be made by a user with no knowledge of the language used to program the system.
Extendable	:	The programmer can easily enlarge the system to encompass a wider range of the data analysis.
Open	:	The user, rather than the program, initiates and controls the interaction and accordingly gains ultimate control of the processing sequence.

Figure 2. Design Considerations for PAS-II.

Theoretical Substructure

Problem Space. We assume human problem solving takes place by search in a problem space. The elements of this space are the possible states of knowledge the subject can have about the task, where a state of knowledge is simply an expression of what the subject knows at some particular point in the space. Besides knowledge states, the problem space also includes a set of operators. These define operations the subject can perform on knowledge at a particular state to yield new knowledge -- hence to move to a new knowledge state. The operators are incremental, that is, they take as input a small portion of the total knowledge state (a small set of knowledge elements) and produce as output new knowledge elements.

Problem Behavior Graph. The subject's search through the problem space for a solution can be described as a sequence of operator applications that create a string of incrementally changing knowledge states. The plot of this search is called the problem behavior graph (PBG). Figure 8 (also used to illustrate the output of the analysis given in Section 4) shows a problem behavior graph for cryptarithmetic. The nodes represent operator applications: the knowledge elements at the lower left of each node are the inputs, those at the lower right are the outputs. PBG branching results from the subject abandoning information and returning to a prior knowledge state (usually because of a discovered contradiction). For example, in Figure 8 the outputs of nodes 4 and 6 conflict: "R is 4" conflicts with "R is odd," and leads to the abandonment of nodes 4, 5 and 6. Note that the knowledge state at any point in the graph is the conjunction of all output elements on the path from the given point back to the beginning of the graph. All nodes on the path from the last node back to the beginning of the graph are called currently active nodes. Their output elements define the current knowledge state.

Data Analysis

The data being analyzed is the transcribed text of a subject's verbal protocol. As the text is transformed into a PBG it is subjected to four major types of processing: linguistic, semantic, group, and PBG. Figure 1 typifies such a processing sequence.

Linguistic Processing. The text is first segmented into shorter strings called topic segments, each of which is expected to ultimately yield approximately one problem space element. Each segment is then parsed using a grammar sensitive to the problem domain under consideration. The result of parsing is a set of semantic elements which represent the meaning of the segment. For example, the segment "D is not equal to 6" might yield the elements (NEG)(EQ D 6) in the cryptarithmetic task. Here (NEG) is called an indicator element, (EQ D 6) a knowledge element.

Semantic Processing. The semantic elements produced through parsing are first combined in very elementary ways to produce new elements, i.e., (NEG) and (EQ D 6) become (NEQ D 6). Next, new elements reflecting relationships between elements from adjacent segments are produced. Thus, (EQ D 5) from one segment and (THEREFORE)(EQ T 0) from the next segment become (BECAUSEOF (EQ D 5)(EQ T 0)), e.g., "because D is 5, T is 0." Finally, these elements are arranged into initial approximations of operator groups, each containing an operator element and the surrounding knowledge and indicator elements. An operator group is defined to be an operator together with its input and output knowledge elements.

Group Processing. The tentative operator groups produced during semantic processing are now analyzed to obtain a complete picture of what the subject knows at each moment and what operators he applies. First, variables in semantic elements are identified by comparing the elements to the current context as defined by the PBG. Thus if (EQ D 5) were in the PBG then when given the element (EQ <L> 5), where <L> stands for a class of letters, we recognize that <L> in this case is the letter D.

The second part of group processing consists of finding, or hypothesizing, the origin of every knowledge element in each tentative group. The origin of a knowledge element is defined to be the operator which produced it, plus the inputs to that operator, plus the operators which produced those inputs, etc. Thus the origin can be represented as a tree which defines a collection of overlapping operator groups.

PBG Processing. The operator groups produced during group processing are now incorporated into the PBG. In general, each group becomes a node in the PBG. In the simplest case the new node is just attached to the last currently active node. However, when contradictions occur (the output of one node conflicts with the output of another) restructuring occurs to eliminate the conflict (see Figure 8).

3. Structure of the Program

PAS-II takes as input a transcribed text of the verbalization of a subject solving a problem and produces as output a PBG. The processing rules for the various stages, including the rules defining the problem space, are given to the system. These rules are supplied either by the system builder via a library of rules for various problem domains or by the user himself.

Modular Structure

PAS-II is organized as a modular data analysis system. The basic unit of organization is the mode: a processing state which has associated with it a buffer capable of holding rules or data. This buffer can be modified by the editing functions available in the command language. There are three types of modes: run modes, which hold the data being analyzed, rule modes, which hold the processing rules, and auxiliary modes, which hold task-free system-oriented rules. Thus the information in the rule modes constitutes the problem dependent part of the system.

The next level of organization is the stage: a unit consisting of one run mode and any number of associated rule modes. Data processing is performed in a stage by applying the rules from the rule modes associated with that stage to the data present in the run mode of the previous stage. The result of the processing is then put into the run mode of the current stage. Figure 3 illustrates the modular organization of PAS-II, with the arrows indicating data flow and the lines indicating mode associations.

The highest level of organization is the processor: a unit consisting of consecutive stages in the control cycle. For example, in PAS-II two linguistic stages form the Linguistic processor and three semantic stages form the Semantic processor.

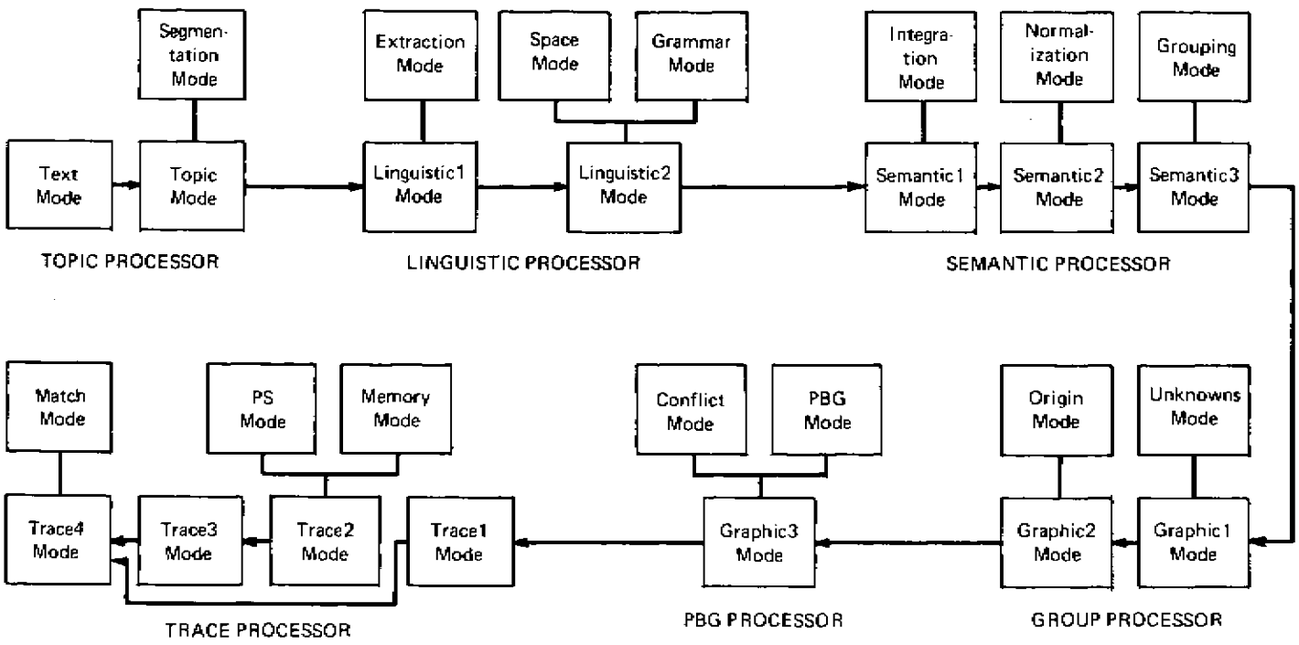

Figure 3. Modular organization of PAS-II
(Auxiliary modes not shown)

Modes. The modes currently implemented in PAS-II are listed in Table 1. Note that most run modes have one or two rules modes associated with them. This association is illustrated in Table 1 and also in Figure 3," which shows the modular composition of the various processors in PAS-II. The arrows in the figure define the data links existing between modes. The mode at the tail of an arrow provides the data that the mode at the head of the arrow processes. For example, processing in the TOPIC mode involves applying the SEGMENTATION rules to the data in the TEXT mode and then placing the result in the TOPIC mode. As each line in TEXT is processed, it is deleted from the TEXT buffer. However, a copy of these deleted lines is stored elsewhere in TEXT and can be retrieved (see the process functions in Table 2). The arrows in Figure 3 do not necessarily define the control cycle, i.e., the order in which processing occurs. The control flow is illustrated in Figure 4 (to be discussed later).

MODES		
RUN	RULE	AUXILIARY
TEXT		ASSOCIATION
TOPIC	SEGMENTATION	SAVE
LINGUISTIC1	EXTRACTION	CONTROL
LINGUISTIC2	SPACE, GRAMMAR	INFORMATION
SEMANTIC1	INTEGRATION	
SEMANTIC2	NORMALIZATION	
SEMANTIC3	GROUPING	
GRAPHIC1	UNKNOWNS	
GRAPHIC2	ORIGIN	
GRAPHIC3	CONFLICT, PBG	
TRACE1		
TRACE2	PS, MEMORY	
TRACE3		
TRACE4	MATCH	

Table 1. PAS-II Modes.

Functions. The functions currently implemented in PAS-II are listed in Table 2. They constitute the command language available to the user, and are divided into four categories: basic, edit, flag, and process functions. Note that a mode name is a function that puts the user into that mode.

A function call consists of a function name followed by its arguments. Any number of function calls may occur together. If it is not clear which names are the functions and which are the arguments, parentheses can be used for disambiguation. In ambiguous cases the system always assumes the name is a function name rather than an argument. Thus if the user types HELP TOPIC DISPLAY 3 it could mean either (HELP TOPIC): give me information about the TOPIC mode, and (DISPLAY 3): display line 3 of the current buffer; or (HELP): tell me how to get help, (TOPIC): put me into the TOPIC mode, and (DISPLAY 3): display line 3. The system would make the latter interpretation.

Comparison with Figure 1 shows how PAS-II maps onto PAS-I. Note that the scope of the analysis has been extended to include a Trace processor (not discussed in detail in this paper).

Auxiliary Modes. There are four auxiliary modes: save, control, association, and information. The SAVE mode contains rules which specify which mode buffers are to be saved on (or read into from) a disk file when the WRITE (or READ) command is executed. The CONTROL mode contains rules which define the control cycle for the system. Initially these rules define the control flow shown in Figures 3 and 4. The ASSOCIATION mode contains rules which define the associations between run and rule modes. The initial (or default) associations are those shown in Figure 3. The CONTROL and ASSOCIATION modes, together with the CREATE function, permit the sophisticated user to create new modes, redefine mode associations, and reorganize the control flow for the entire system. One example of this is the use of a reorganized PAS-II to analyze a problem description (problem text) in natural language in order to infer from that text a tentative problem space, one that a subject might use in representing the problem (2).

The INFORMATION mode is unique in containing no buffer and recognizing none of the functions that constitute the command language. Instead, this mode responds to key words in the users input, which may be in sentence form. The mode provides the user with general information about PAS-II: its basic organization, purpose, and techniques of operation. This is to be contrasted with the HELP function, which provides the user with specific, on-the-spot information about the mode he is in.

Control Structure

The control cycle for PAS-II is shown in the flow diagram of Figure 4. The solid arrows indicate the stage that is entered once processing in the current stage is finished. The broken arrows indicate which stage to enter before processing is started. Processing in LINGUISTIC1, SEMANTIC3, and GRAPHIC2 is incremental. In each of these modes only part of the data from the previous mode is processed at one time. This initial portion of the data is then carried through the rest of the system, leading to the growth of PBG nodes, before the rest of the data in the previous mode is processed. This is done to establish a semantic context (the PBG) as early as possible in the processing sequence so it can provide feedback needed for linguistic, semantic, and group processing'.'

Since the control organization of PAS-II is quite flexible, the user is under no constraints to process the data in the order shown in Figure 4. He may skip or repeat stages within the existing control framework, and may redefine the control cycle (via the CONTROL mode). He may also have the system put him into the next run mode in the control loop, or even automatically step him through the run modes, initiating the processing at each stage (see NEXT and AUTOMATIC in Table 2).

Data Processing

Figures 3 and 4 show the processors which comprise the control cycle of PAS-II. In the Topic processor transcribed text is segmented into phrases containing only a single task topic.** Then in the Linguistic processor an initial collection of these

At present the PBG provides feedback for group processing only.

This is a slight extension: PAS-I requires segmented text as input.

	FUNCTIONS	
	NAME	DESCRIPTION
	(mode name)	Puts user into the mode named.
	CREATE	Creates a new mode.
	DISPLAY	Displays the contents of M.
B	ERASE	Uncreates M (if it was formed using CREATE).
A	EXIT	Takes the user out of the system (to LISP).
S	HELP	Provides system information pertinent to M.
I	MODE	Tells the user what mode he is in.
C	NEXT	Puts the user into the next appropriate run mode of C.
	RULE	Puts the user into the rule mode associated with M.
	RUN	Puts the user into the run mode associated with M.
	BREAK	Breaks a line in M into two or more smaller lines.
	CONNECT	Connects adjacent lines in M to form a single line.
E	DEFINE	Permits the user to define the contents of lines in M.
D	DELETE	Deletes lines in M.
I	ED	Enables the user to perform intra-line editing in M.
T	INSERT	Inserts a line after a given line in M.
	READ	Reads data from a disk file into M.
	RENUMBER	Renumbers the lines in M.
	WRITE	Write the contents of M onto a disk file.
	AUTOMATIC	Steps the user through C, executing GO in each run mode.
	BATCH	Stops system queries during run mode processing.
	COMMENT	Permits comments to be displayed when a line is displayed.
F	FAST	Speeds up reading from the disk by eliminating format checking.
L	HUSH	Abbreviates error messages.
A	NUMBERS	Causes disk files to be written with buffer line numbers.
C	PRINT	Puts all the I/O at the terminal onto a disk file.
	SEARCH	Causes processing to be repeated until no rules are applicable.
	SUPPRESS	Suppresses printing of auxiliary information during processing.
	TIME	Causes processing time in M to be printed.
	VERSION1	Causes the old version of grammar/parser to be used.
	VERSION2	Causes the new improved version of grammar/parser to be used.
P	AGAIN	Puts the data in M into P and fires GO.
R	COPY	Prints the copy of the data in M.
O	GO	Processes the data located in P and puts the result into M.
C	RECOPY	Puts the copy of the data from M back into M.
E	RESTART	Puts the copy of the data from P back into P and fires START.
S	START	Deletes the data in M and fires GO.
S		

KEY M: mode buffer of the mode the user is in
P: mode buffer prior to M in C
C: control cycle

Table 2. Description of PAS-II Functions
(Flag descriptions are for the condition flag T)

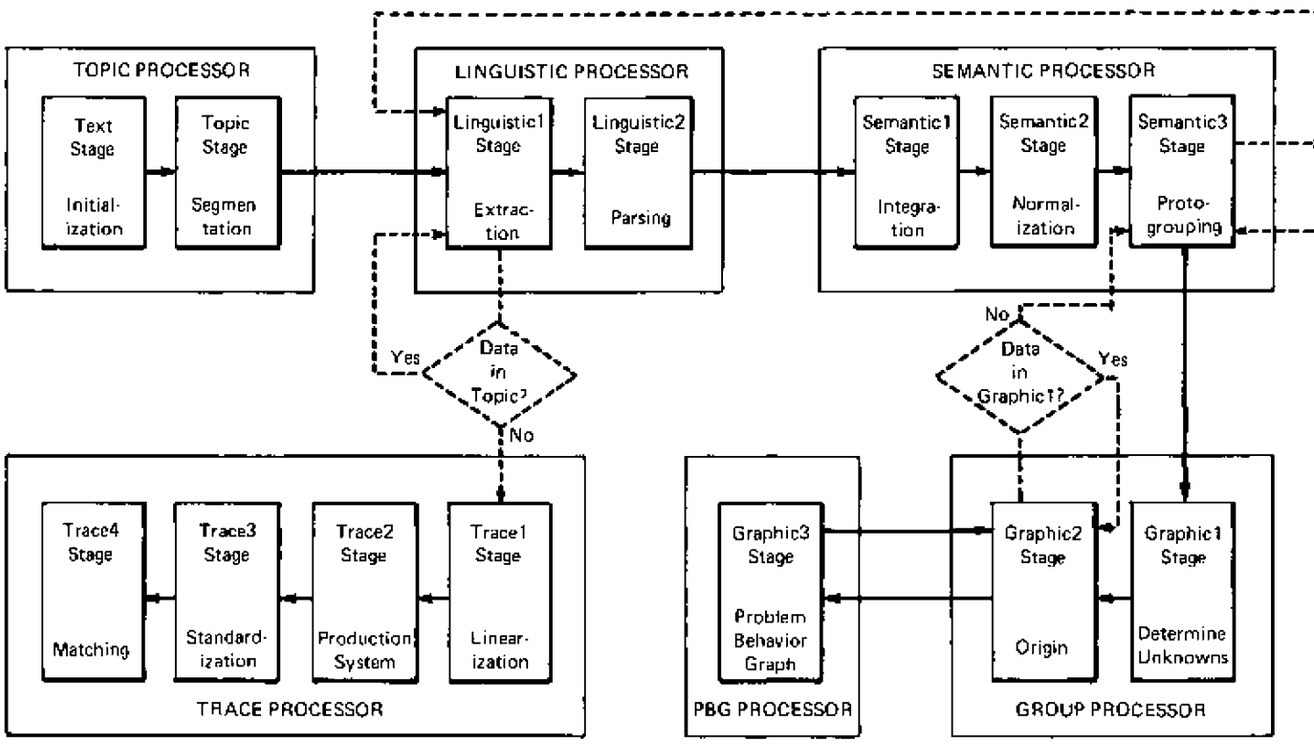

Figure 4. Flow diagram of PAS-II

Key: ——→ stage to enter after processing

- - - → stage to enter before processing

segments is parsed yielding sets of semantic elements. These elements are processed and refined in the Semantic processor to produce groups composed of one operator element and its associated input and output knowledge elements. In the PBG processor these groups are incorporated into the PBG. The Trace processor is then used to compare this PBG with the trace produced by a given production system model of the subject.

Topic Processor. The Topic processor contains two run modes: TEXT and TOPIC. TEXT is an initialization mode; it holds the data for TOPIC to process. Thus no real processing takes place in it. The TOPIC mode uses the SEGMENTATION rules to segment all the text in the TEXT mode. These rules have the general form: $\underline{\text{string}}_1$ / $\underline{\text{string}}_2$, where a string is any sequence of words, punctuation marks, or word classes (as defined in the GRAMMAR mode), including the null sequence. The slash (/) indicates where the text is to be broken, i.e., after every occurrence of string_1 that is immediately followed by an occurrence of string_2. Figure 6 show SEGMENTATION rules for cryptarithmetic (to be used in the example in Section 4).

Linguistic Processor. The Linguistic processor contains two run modes: LINGUISTIC1 and LINGUISTIC2. In LINGUISTIC1 the EXTRACTION rules are used to select a consecutive set of segments from TOPIC, representing an initial guess as to the minimum number of segments from which a group can be inferred. Processing consists only of transferring these segments from the TOPIC mode to the LINGUISTIC1 mode. At present, the EXTRACTION rules are simply a single integer specifying how many segments to transfer.

Processing in the LINGUISTIC2 mode consists of applying the SPACE and GRAMMAR rules to all the topic segments in LINGUISTIC1. The parsing operation produces, for each segment, a set of semantic elements representing the meaning of the segment. The rules in the SPACE mode define the problem space and have the form:* (semantic-element) type, where a semantic element is either an operator, knowledge, or indicator element, and the type is either OP, KN, or IND. The GRAMMAR** rules define a key-word grammar and have the form: <class> = (item_{11} item_{12} ...) (item_{21} item_{22} ...) ... , where an item is either a class (denoted by angle brackets) or a literal (such as a word, letter, or character). An asterisk (*) can be used between any two items to indicate a match with any string of text, and any GRAMMAR rule which is a disjunction of single literals can be written without parentheses. Figure 6 shows SPACE and GRAMMAR rules for cryptarithmetic.

Semantic Processor. The Semantic processor contains three run modes: SEMANTIC1, SEMANTIC2, and SEMANTIC3. In SEMANTIC1 the INTEGRATION rules produce new elements by combining semantic elements generated from the same or adjacent segments. In SEMANTIC2 the NORMALIZATION rules map knowledge and indicator elements into single elements reflecting the relationships existing between two or more knowledge elements. In SEMANTIC3 a tentative operator group (protogroup) is formed. The INTEGRATION AND NORMALIZATION rules are replacement rules of the type A => B, i.e., replace A with B. Both A and B can be lists of semantic elements. A slash (/) indicates that the next elements of the list occur on the next line of the mode buffer. Class names and X's are used as variables, and in the NORMALIZATION rules A's are variables which stand for knowledge elements on adjacent lines connected by the AND indicator. Typical INTEGRATION and NORMALIZATION rules for cryptarithmetic are shown in Figure 6. GROUPING rules are not shown.* They define a protogroup to be the largest consecutive sequence of elements containing no more than one operator element.

Group Processor. There are two run modes in the Group processor: GRAPHIC1, and GRAPHIC2. GRAPHIC1 processing fills in the values of variables in the semantic elements by comparing the element containing variables with all the elements currently active in the PBG, i.e., the current context. When a match is found the appropriate values are filled in. Currently the UNKNOWNS rules are not accessible to the user.

Processing in GRAPHIC2 is a joint man-machine effort.** The goal is to hypothesize for each knowledge element its origin, i.e., the operator and its inputs (and the operators that produced those inputs, etc.) that produced that knowledge element as output. The system queries the user asking for possible operators and inputs that could have produced the element whose origin is being sought. From this information the system constructs an origin tree, and hypothesizes which path through the tree represents the actual origin of the element. The path is picked on the basis of the agreement between the hypothesized inputs and the actual context defined by the current PBG. The ORIGIN rules, like the GROUPING and UNKNOWNS rules, are currently not accessible.

PBG Processor. The PBG processor contains one run mode: GRAPHIC3. In the GRAPHIC3 mode, processing consists of taking the operator groups produced in GRAPHIC2 and incorporating them into the problem behavior graph. The CONFLICT rules are used to determine whether or not any knowledge elements in the operator groups conflict with knowledge already in the PBG. If such a conflict occurs, the PBG rules are used to restructure the PBG so the conflict is eliminated.

---

* SPACE rule 8 in Figure 6 is an exception. It defines a set named <V> containing two members, the class <LETTER> and the class <CARRY>.

** Two parsers are available, a simple top down parser and a more sophisticated parser written by M. Rychener.

* At the current stage of development the Grouping rules have not been made accessible to the user.

** This is the major place where we have not regained in PAS-II the power for automatic processing available in PAS-I.

Both the CONFLICT and PBG rules are ordered production rules of the form S → A, i.e., in situation S take action A (12, 13). A situation is defined by a list of values of certain variables, called the state vector, SV. The left side of each production rule has the form $(V_1\ V_2\ V_3\ \ldots)$, where V_n represents a permissible value for the nth state vector variable. The right side has the form $(A_1\ A_2\ A_3\ \ldots)$, where the A's represent actions to be taken. The current values of the state vector variables are compared with the left side of each production rule. The first match, from top to bottom, determines the actions to be taken (an asterisk is considered to match any value).

Figure 6 shows CONFLICT and PBG rules for cryptarithmetic. The CONFLICT rules determine whether or not two given knowledge elements conflict. The example CONFLICT state vector contains: (SAME 2), which is true (T) if the second items of both the elements are identical and false (F) otherwise; (ITEM 1 1), which returns as a value the first item of the first element (the element in the PBG); and (ITEM 1 2), which returns as a value the first item of the second element (the element in the group). Thus if the two elements being compared were (ODD R) and (NEQ R 5) CONFLICT rule 3 would match the state vector and the decision would be that no conflict exists.

The PBG rules determine the type of restructuring that occurs once a conflict is detected. The PBG state vector in Figure 6 has 2 variables: TYPE, which has the value CON if restructuring is based on conflict and SIM if it is based on similarity;* and (ITEM 1 2), which is defined above. The actions shown in Figure 6 are BLOCKREM, a type of restructuring where blocks of adjacent nodes are abandoned, and COPY, a specification that the group causing the restructuring should remain in the active portion of the PBG after restructuring. The state vectors for CONFLICT and PBG may contain variables and actions other than the ones shown in Figure 6. For a complete description of these rules see the PAS-II reference manual (16).

Trace Processor. The Topic, Linguistic, Semantic, Group and PBG processors comprise the major portion of PAS-II. It is this portion which represents a generalized version of PAS-I. The Trace processor is a new extension to the system and has no analogue in PAS-I. Some parts of it, like the MATCH mode, are still under development. The Trace processor enables the user to write a production system model of the subject (6), and then compare the trace obtained by running the production system model with the PBG obtained by analyzing the protocol. The details are described elsewhere (16).

4. Example of Program Operation

To illustrate the use of PAS-II, we present a listing of the actual user-machine interaction involved in the on-line analysis of a short cryptarithmetic protocol. The cryptarithmetic task is given in Figure 5. Both the protocol and the cryptarithmetic rules used for this example are shown in Figure 6. The protocol is stored in the TEXT mode and the cryptarithmetic rules in the eight rules modes shown. These rules approximate the minimal set needed to analyze the given protocol, and are for expository purposes only.*

The annotated listing is shown below. The user input is in lower case and the system output in upper case. The system prompts the user by typing either an asterisk (*) or a question followed by a question mark (?).

```
*text display
TEXT MODE
     1. D IS 5 ; THEREFORE T IS 8 . ASSUME R EQUALS 4 . SINCE YOU
        CARRY 1 , R IS ODD . ASSUME R IS 7 , NOT 5 .
*next go

TOPIC MODE
1. D IS 5 ;
   THEREFORE T IS 8 .
   ASSUME R EQUALS 4 .
   SINCE YOU CARRY 1 ,
   R IS ODD .
   ASSUME R IS 7 ,
   NOT 5 .
OK? yes
TOPIC MODE FINISHED
*next go

LINGUISTIC1 MODE
   1. D IS 5 ;
   2. THEREFORE T IS 8 .
   3. ASSUME R EQUALS 4 .
   4. SINCE YOU CARRY 1 ,
   5. R IS ODD .
   6. ASSUME R IS 7 ,
   7. NOT 5 .
OK? yes
*next go
```

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

The above expression 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?

Figure 5. Cryptarithmetic Task

The user first entered the TEXT mode and displayed its contents. He then entered the next mode in the control cycle, TOPIC, and started processing by typing GO. This caused the SEGMENTATION rules to be applied to the data in TEXT. The system indicated that the data in line 1 of the previous mode had been transformed into the seven lines shown above, and asked if this transformation was satisfactory (OK?). At this point the user typed yes, telling the system to actually put those seven lines into the next seven

* The PBG rules are also used for restructuring when similarities (identical nodes) are detected, as discussed in an earlier paper on PAS-I (15).

* At least four times as many rules would be needed for a complete set (15).

```
      TEXT MODE
1. D IS 5 ; THEREFORE T IS 0 . ASSUME R EQUALS 4 . SINCE YOU
   CARRY 1 , R IS ODD . ASSUME R IS 7 , NOT 5 .

      SPACE RULES
1. (NEG) IND
2. (ODD <V>) KN
3. (EQ <V> <DIGIT>) KN
4. (THEREFORE) IND
5. (BECAUSE) IND
6. (ASSUME) IND
7. (DIGIT <DIGIT>) KN
8. (<V> <LETTER> <CARRY>) SPASET

      GRAMMAR RULES
1. <EQ> = (<CARRYEQ>) (<LETTER> * <EQUAL> * <DIGIT>)
2. <CARRYEQ> = (<CARRY> * <DIGIT>) (<CARRY>)
3. <ODD> = (<LETTER> * <EQUAL> * ODD)
4. <EQUAL> = IS EQUAL EQUALS BE WAS ARE
5. <NEG> = CANNOT NOT NO N'T
6. <THEREFORE> = THEREFORE IMPLIES
7. <ASSUME> = ASSUME ASSUMING
8. <BECAUSE> = BECAUSE SINCE
9. <CARRY> = CARRY CARRYING CARRIED
10. <LETTER> = A B D E G L N O R T
11. <DIGIT> = 0 1 2 3 4 5 6 7 8 9

      SEGMENTATION RULES
1. . /
2. ; /
3. <DIGIT> , /
4. <LETTER> , /

      EXTRACTION RULES
1. 12

      INTEGRATION RULES
1. (X1 CARRY X2)  =>  (X1 <C> X2)
2. (EQ X1 X2) / (DIGIT X3)  =>  (EQ X1 X2) / (EQ X1 X3)
3. (NEG) (EQ <LETTER> <DIGIT>)  =>  (NEQ <LETTER> <DIGIT>)
4. (ASSUME) (EQ <LETTER> <DIGIT>)  =>  (AEQ <LETTER> <DIGIT>)

      NORMALIZATION RULES
1. A1 / (THEREFORE) A2  =>  (BECAUSEOF A1 A2)
2. (BECAUSE) A1 / A2  =>  (BECAUSEOF A1 A2)

      CONFLICT RULES
1. SV = ((SAME 2) (ITEM 1 1) (ITEM 1 2))
2. (F * *) => NO-CON
3. (* ODD NEQ) => NO-CON
4. (* * *) => ASK-IF-CON

      PBG RULES
1. SV = (TYPE (ITEM 1 2))
2. (CON NEQ) => BLOCKREJ
3. (CON *) => (BLOCKREJ COPY)
4. (* *) => BLOCKREJ
```

Figure 6. Cryptarithmetic Rules.

lines of the TOPIC buffer. If the processing had been unsatisfactory, the user could have jumped to the SEGMENTATION mode, changed the rules, jumped back to TOPIC, and reprocessed the data using the new rules before proceeding with the next processing step.

The user then entered the next mode, LINGUISTIC1, and started processing. The EXTRACTION rules were applied to the seven lines of data in TOPIC and the system indicated that the processing should consist of placing these lines in LINGUISTIC1 unchanged. Note that the system indicated that line 1 from TOPIC was transformed into a single line in LINGUISTIC1, etc., as opposed to the previous step where one line in TEXT was transformed into seven lines in TOPIC.

```
LINGUISTIC2 MODE
<EQ>  <LETTER>  D
      <EQUAL>  IS
      <DIGIT>  5
   1. (EQ D 5)
      FROM :   D IS 5 ;
OK? yes batch suppress
BATCH=T
SUPPRESS=T
   2. (EQ T 0) (THEREFORE)
      FROM :   THEREFORE T IS 0 .
   3. (EQ R 4) (ASSUME)
      FROM :   ASSUME R EQUALS 4 .
   4. (EQ CARRY 1) (BECAUSE)
      FROM :   SINCE YOU CARRY 1 ,
   5. (ODD R)
      FROM :   R IS ODD .
   6. (EQ R 7) (ASSUME)
      FROM :   ASSUME R IS 7 ,
   7. (NEG) (DIGIT 5)
      FROM :   NOT 5 .
LINGUISTIC2 MODE FINISHED
*(batch f)(suppress f) automatic
BATCH=F
SUPPRESS=F
AUTOMATIC=T
*next go
```

Processing in LINGUISTIC2 consisted of applying the SPACE and GRAMMAR rules to the data in LINGUISTIC1 to produce a parse. In step 1 the parse tree was printed and the user set the flag BATCH true to eliminate the OK? question (the system then assumes the answer is always yes) and the flag SUPPRESS true to eliminate further printing of the parse trees. Then, before going to the next mode in the control cycle, the user set the flag AUTOMATIC true so the system would automatically step through the appropriate run modes executing GO. At this point the LINGUISTIC2 buffer held the seven sets of semantic elements shown above.

```
SEMANTIC1 MODE
RULES APPLIED :   4  1  2  4  3
   1. (EQ D 5)
   2. (EQ T 0) (THEREFORE)
   3. (AEQ R 4)
   4. (BECAUSE) (EQ <C> 1)
   5. (ODD R)
   6. (AEQ R 7)
   7. (NEQ R 5)
OK? yes
SEMANTIC1 MODE FINISHED
```

```
SEMANTIC2 MODE
RULES APPLIED :   1  2
1-7. (BECAUSEOF ((EQ D 5)) ((EQ T 0)))
     (AEQ R 4)
     (BECAUSEOF ((EQ <C> 1)) ((ODD R)))
     (AEQ R 7)
     (NEQ R 5)
OK? yes
SEMANTIC2 MODE FINISHED

SEMANTIC3 MODE
   1. (BECAUSEOF ((EQ D 5)) ((EQ T 0)))
   2. (AEQ R 4)
   3. (BECAUSEOF ((EQ <C> 1)) ((ODD R)))
   4. (AEQ R 7)
   5. (NEQ R 5)
OK? yes
```

Processing in SEMANTIC1 consisted of applying the INTEGRATION rules to the semantic elements in LINGUISTIC2. As indicated above there were five applications of the rules. Processing in SEMANTIC2 consisted of applying the NORMALIZATION rules to the seven sets of elements in SEMANTIC1. There were two applications of the rules, and five sets of elements were left in SEMANTIC2. Processing in SEMANTIC3 consisted of applying the GROUPING rules, which are not explicit. These rules simply attempted to pull from SEMANTIC2 one operator element and its associated knowledge elements. Since no operator elements were present, it pulled all the elements from SEMANTIC2.

```
GRAPHIC1 MODE
   1. (BECAUSEOF ((EQ D 5)) ((EQ T 0)))
      FROM :   (BECAUSEOF ((EQ D 5)) ((EQ T 0)))
OK? yes
   2. (AEQ R 4)
      FROM :   (AEQ R 4)
OK? yes
   3. (BECAUSEOF ((EQ <C> 1)) ((ODD R)))
      FROM :   (BECAUSEOF ((EQ <C> 1)) ((ODD R)))
OK? yes batch suppress r: (becauseof ((eq c2 1))((odd r)))
BATCH=T
DO YOU REALLY WANT BOTH AUTOMATIC=T AND BATCH=T ? yes
SUPPRESS=T
   4. (AEQ R 7)
      FROM :   (AEQ R 7)
   5. (NEQ R 5)
      FROM :   (NEQ R 5)
GRAPHIC1 MODE FINISHED
```

Processing in GRAPHIC1 consisted of applying the UNKNOWNS rules, which are not explicit. These rules involve searching the existing PBG for elements that match the elements containing unknowns. In this simple example no matches were found because the PBG had not yet been grown. Thus, in step 3 when the unknown carry <C> was not found, the user told the system to replace its processing result with (BECAUSEOF ((EQ C2 1)) ((ODD R))). This was put into line 3 of the GRAPHIC1 buffer, rather than the result containing <C>. In effect the user told the system that the value of <C> was C2, i.e., that the unknown carry was the carry into the second column (the L+L=R column).

Processing in GRAPHIC2 and GRAPHIC3 occurred as follows: GRAPHIC2 was entered and the elements from line 1 of GRAPHIC1 were processed interactively to determine their operator groups. GRAPHIC3 was then entered and these groups were grown as new nodes in the PBG. Next GRAPHIC2 was reentered and the elements

from line 2 of GRAPHIC1 processed. This graphic2-graphic3 loop was repeated for each line in GRAPHIC1. Below is shown only one of these loops*: processing and growing the elements from line 3 of GRAPHIC1.

```
GRAPHIC2 MODE
FOR (BECAUSEOF ((EQ C2 1)) ((ODD R))) :
OP = (pc 2)
OUTPUTS = (odd r)
INPUTS = (eq c2 1)
FOR (EQ C2 1) :
OP = (av c2)
INPUTS =
OTHER ORIGINS FOR (EQ C2 1) ? yes
FOR (EQ C2 1) :
OP = (pc 1)
INPUTS = (eq d 5)(eq c1 0)
(EQ D 5) FOUND IN PBG
(EQ C1 0) FOUND IN PBG
OTHER ORIGINS FOR (EQ C2 1) ? no
ORIGIN TREE :
(ODD R)  (PC 2)  (EQ C2 1)  (AV C2)
                            (PC 1)  (EQ D 5)
                                    (EQ C1 0)
3. (PC 1) ((EQ D 5) (EQ C1 0)) (EQ C2 1)
   (PC 2) ((EQ C2 1)) (ODD R)
      FROM :   (BECAUSEOF ((EQ C2 1)) ((ODD R)))

GRAPHIC3 MODE
   1. GROW (EQ C2 1)
      FROM :   (PC 1) ((EQ D 5) (EQ C1 0)) (EQ C2 1)
DO (AEQ R 4) AND (ODD R) CONFLICT ? yes
   2. CONFLICT: N4 (AEQ R 4) AND (ODD R) WITH (BLOCKREJ COPY)
      FROM :   (PC 2) ((EQ C2 1)) (ODD R)
GRAPHIC3 MODE FINISHED
```

In GRAPHIC2 the system queried the user to determine possible origins (operators and their inputs) for the elements in question. This information was represented as an origin tree as shown above. This tree is displayed below in a more conventional style.

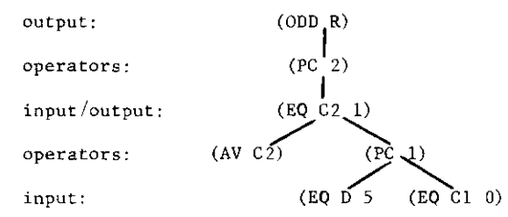

Figure 7. Origin Tree

The system analyzes the tree and decides which path represents the best origin for the top element, in this case (ODD R). Here there are only two alternatives: the path with the operator: assign a value to the carry into column 2, (AV C2), and the path with the operator: process column 1, (PC 1). The system chooses the latter, based on implicit ORIGIN rules which tell it to choose between operators by rating them according to their inputs. The decision function currently in use is:

Choose to maximize: (3 x used-inputs) - (unused-inputs)

where an input is "used" if it occurs in the PBG. Thus (AV C2) has a rating of 0 while (PC 1) has a rating of (3x2)-0 or 6. The format of the operator groups produced in GRAPHIC2 is: operator (input list) output.

In GRAPHIC3 the two groups from GRAPHIC2 were incorporated into the PBG. The second group, with (ODD R) as the output, conflicted with an existing group in the PBG and led to restructuring of the PBG to resolve the conflict. Conflicts were defined by the CONFLICT rules, the type of restructuring by the PBG rules.*

```
*graphic3 display
GRAPHIC3 MODE
     N1      0   OP (RECALL D)   OUT (EQ D 5)
     N2          OP (RECALL C1)   OUT (EQ C1 0)
     N3          OP (PC 1)   IN (EQ D 5) (EQ C1 0)   OUT (EQ T 0)
     N4          OP (AV R)   OUT (AEQ R 4)
     N5          OP (PC 1)   IN (EQ D 5) (EQ C1 0)   OUT (EQ C2 1)
     N6          OP (PC 2)   IN (EQ C2 1)   OUT (ODD R)     0
     N7      3   OP (PC 1)   IN (EQ D 5) (EQ C1 0)   OUT (EQ C2 1)
     N8          OP (PC 2)   IN (EQ C2 1)   OUT (ODD R)
     N9          OP (AV R)   OUT (REQ R 7)
    N10          OP (TD R 5)   IN (EQ D 5)   OUT (NEQ R 5)
```

After all the data from GRAPHIC1 was processed in GRAPHIC2 and GRAPHIC3 the contents of GRAPHIC3 were displayed. Each line in the display represents a node in the PBG. Node 10 contains the operator: test to see if R can have the digit 5 as a value, (TD R 5). Figure 8 shows this PBG in the conventional representation. Note that the conflict between (AEQ R 4) and (ODD R) led to a back-up that abandoned nodes 4, 5 and 6. Thus the currently active nodes, the ones that define the current context, are those joined by the heavy lines in Figure 8.

5. Discussion

The initial program, PAS-I, is an artificial intelligence program by any reasonable criteria. The task it attempts, the inference from verbal behavior to Problem Behavior Graph, is a task requiring intelligence when done by humans. The mechanisms used are those common to other artificial intelligence programs that tackle somewhat similar tasks: grammars to deal with the surface structure of natural language, representation of knowledge, matching, and heuristic search to infer information not directly expressed in the utterances.

PAS-II is a program that accomplishes the same task as PAS-I. Hence, it too is an artificial intelligence program. But when looked at structurally it more closely resembles a data processing framework or, possibly, a language. Something has happened in going from PAS-I to PAS-II, something worth identifying and discussing.

Let us start with Planner (3) and QA4 (8).** These systems are languages for writing programs to perform a class of artificial intelligence tasks. The

---

* Space limitations prevent us from including the entire listing.

* Conflict and PBG rules are described in detail in an earlier paper (15).

** There are other representatives of this class, e.g., POPLER (1) and Conniver (10, 11).

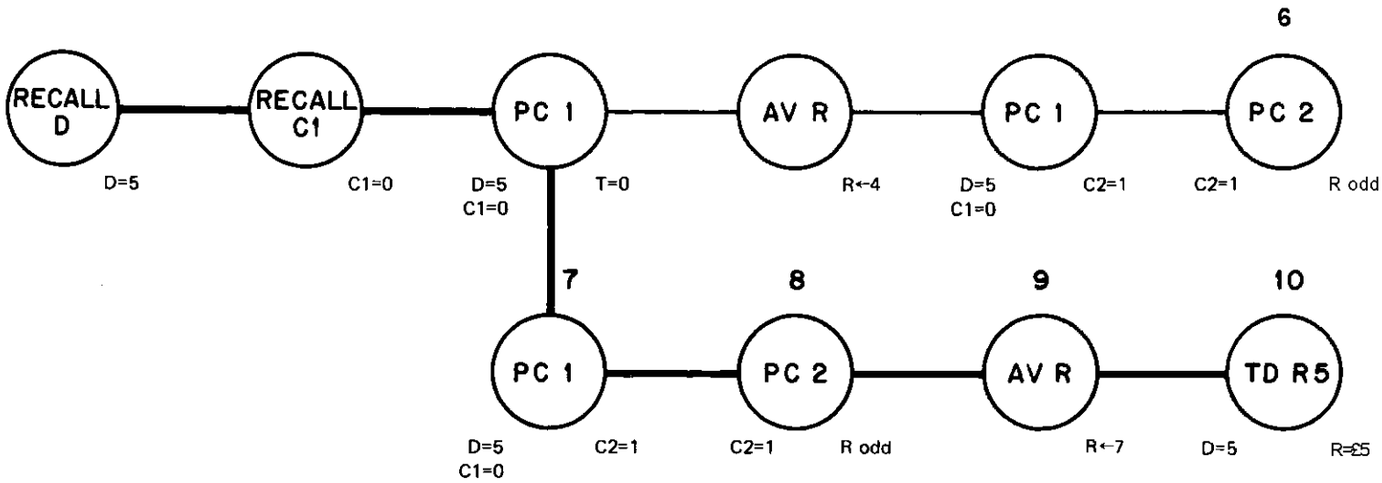

Figure 8. Problem Behavior Graph for Cryptarithmetic

		Knowledge		*Operators*
		equal	RECALL	recall element
KEY:	«-	assign equal	PC	process column
		not equal	AV	assign value
			TD	test digit

exact boundaries of these tasks are obscure but their central core is clear and includes a large fraction of the tasks for which heuristic programs have been built -- theorem proving, robot planning, symbolic manipulation, etc. These systems were formed, essentially, by taking a list processing framework and embedding within it some of the ad hoc mechanisms developed for particular heuristic programs. They include backtracking, a generalized matching facility, a global data base (accessed by pattern matching) and multiprocessing control. Embedding these mechanisms within a language makes possible their use in novel combinations (and in interaction with the other mechanisms available in higher languages).

This same embedding of mechanisms into a language system has occurred in the transition from PAS-I to PAS-II. PAS-II provides a framework within which a class of AI programs can be easily constructed. This class is not the same as that of the Planner/QA4 type system, which is more "mainline" artificial intelligence. Rather, it appears to be characterized as linguistic data processing, the essential feature being the processing of long sequences of data (rather than just a sentence at a time). This class includes, of course, protocol analysis. It also includes a number of other tasks: content analysis of more classical varieties (9), problem space construction (2), test grading, and what is coming to be called semantic filtering.

The embodiment of mechanisms into a language framework has occurred at two levels in PAS-II, one corresponding roughly to that of Planner/QA4 and the other more specialized. The first level is represented by the PAS-II framework of run modes, rule modes, common command language, editing system, and control structure. This includes a set of mechanisms for the data base (the run modes), a matching facility (the common mechanism for how the rules work on data), and a backtrack facility (the saving of buffers so that processing can be undone). Added to this is the explicit control structure for processing within a stage and passing through the stages, which corresponds to a weak method (4) in the same sense as GPS's basic methods or the basic methods built into the goal construct in Planner/QA4. These provide a schema of operation which, though almost content free, is still a rational procedure for achieving the overall goal. The mechanisms adopted in PAS-II are somewhat more shaped than their correspondents in Planner/QA4, e.g., there is not a single global data base or one stratified by a general context mechanism, rather the data is organized into homogeneous groups (the modes) along structural lines.

The second level is the specialization of the various modes to specific subtasks inherent in tasks of the class: segmentation, parsing, normalization, etc. The specialized rule systems contain the knowledge about the processing. Thus writing any sort of legal rules within a given rule system generates processing of the right sort (though it may not do the right task). In this respect providing a single generalized rule system or scheme for pattern matching and pattern evoked actions (in the manner of Planner/QA4) would move more of the knowledge required back across the boundary from the language system (PAS-II) to the coding within the system (the user program in PAS-II, which is the set of actual rules in the rule modes).

As one moves PAS-II in the direction of a generalized system for a wider class of problems, one can expect the collection of rule modes to increase, becoming eventually, a library in the classic subroutine library sense. The system designer is then faced with the problem of providing these modes with the rules needed to define processing in the various problem domains. However, one advantage of specialized rule systems is that when their structure is highly constrained it becomes easy to predict the effect of modifying rules in the system (as compared to predicting the effect of modifying statements in a general programming language). This sets the stage for the development of self-modifying systems which rewrite their own rules or, in effect, learn to improve their performance in some data processing task (12, 13). Such a capability in an interactive PAS-II-like system would enable the system to build or modify its own rules for a particular problem domain, using feedback from the user to direct the search for good sets of rules.

The evolution from PAS-I to PAS-II in analogy to the more general evolution going on toward planner-like language systems should add to the awareness that embedding mechanisms in language remains a potent scheme for making advances in artificial intelligence.

Acknowledgments

This paper will appear in the preprints for the third International Joint Conference on Artificial Intelligence (IJCAI-73). This research was supported in part by Research Grant MH-07732 from the National Institutes of Health and in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-70-C-0107) which is monitored by the Air Force Office of Scientific Research.

References

1. Davies, D. J. M., POPLER: a POP-2 planner. MIP. School of University of Edinburgh.

2. Hayes, J. R., and Waterman, D. A., Automatic Problem Space Construction, Psychology Department, Carnegie-Mellon University, 1973.

3. Hewitt, Carl, Description and theoretical analysis of planner: A language for proving theorems and manipulating models in a robot. AI report TR-258 (Ph.D. thesis). MIT AI Laboratory, Cambridge, Massachusetts, 1972.

4. Newell, A., Heuristic programming: Ill-structured problems, in Aronofsky, J. S. (ed.) Progress in Operations Research, vol. 3, Wiley, 1969, pp. 362-414.

5. Newell, A., and Simon, H. A., Human Problem Solving, Prentice-Hall, Englewood Cliffs, N.J. 1972.

6. Newell, A., A theoretical exploration of mechanisms for coding the stimulus, in Melton, A. W., and Martin, E. (eds.) Coding Processes in Human Memory, Winston and Sons, Washington, D.C., 1972, pp. 373-434.

7. Newell, A., Simon, H. A., Hayes, R., and Gregg, L., Report on a workshop in new techniques in cognitive research. Computer Science Department, Carnegie-Mellon University, 1972.

8. Rulifson, J. F., Derksen, J. A., and Waldinger, R. J., QA4: A procedural calculus for intuitive reasoning, Stanford Research Institute, November 1972.

9. Stone, P. J., Dunphy, D, C, Smith, M, S,, Ogilvie, D, M,, The General Inquirer, MIT, Cambridge, Massachusetts, 1966.

10. Sussman, Gerald, and McDermott, Drew, Why conniving is better than planning, MIT, Cambridge, Massachusetts, April 1972.

11. Sussman, Gerald, and McDermott, Drew, Conniver Reference Manual, MIT, Cambridge, Massachusetts, May, 1972.

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

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

14. Waterman, D. A., and Newell, A., Protocol analysis as a task for artificial intelligence. Artificial Intelligence, vol. 2, nos. 2 and 3, 1971, pp. 285-318.

15. Waterman, D. A., and Newell, A., Preliminary results with a system for automatic protocol analysis. Carnegie-MelIon University, Computer Science Department, 1973.

16. Waterman, D. A., PAS-II Reference Manual, Psychology Department, Carnegie-Melion University, 1973.