GENERALITY AND GPS

George W. Ernst

and

Allen Newell

Carnegie Institute of Technology
Pittsburgh, Pennsylvania
January, 1967

This work was supported in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-146) and in part by the RAND Corporation, Santa Monica, California.

PREFACE

The material of this monograph constitutes the culmination of work on a single problem solving program, GPS (for General Problem Solver), that stretches back to 1957. The origination of the program was the joint work of J. C. Shaw, H. A. Simon, and A. Newell. During the "middle years", the continuing efforts on reorganization and reprogramming fell to A. Newell; the attempts to exploit the program and its ideas were pursued jointly with H. A. Simon. The results of the last several explorations into modified organizations have never been reported. The final phase, reported here, consists of two intertwined parts. One is the description of the final organization of GPS (i.e., GPS-2-5). This is covered in Chapter III and part of Chapter IV. The responsibility for most of the programming details of this organization must rest with the second author (A. Newell); the first author came on the scene much too late to do anything but be frustrated by them. The second part of this research is the attempt to get this program (perhaps "this thing" expresses our feelings more precisely) to live up to its name, at least marginally. Taking GPS and getting it to be general enough to do a number of different tasks has been the primary responsibility of G. Ernst, and constitutes the substance of his doctoral dissertation at Carnegie Institute of Technology. Chapters II, part of IV, V, and VI cover this research. The writing of the entire document has also been primarily his responsibility.

We have used the term "final" in several places above. This does not indicate any feeling that this document marks a terminus to our research on general problem solvers; quite the contrary is true. However, we do feel that this particular aggregate of IPL-V code should be laid to rest, as having done its part in advancing our understanding of the mechanisms of intelligence.

We would like to acknowledge the continued advice, support and criticism of our colleague, H. A. Simon. Both he and J. C. Shaw were involved in the first several years of research on GPS, and their contribution to the present form of the program is pervasive. In addition, we would like to thank L. W. Gregg and R. W. Floyd who served on George Ernst's Dissertation Committee.

The research reported here has been supported in part by contract SD-146 from the Advanced Research Projects Agency to Carnegie Institute of Technology. It has also been supported by the RAND Corporation, both in the support of A. Newell, first as an employee and more recently as a consultant, and through the award by Carnegie to G. Ernst of the RAND Fellowship in Systems and Communication Sciences.

G. W. Ernst
A. Newell

ABSTRACT

The General Problem Solver (GPS) is a computer program which has been used for explorations into both general mechanisms involved in problem solving and the way that humans solve problems. The program has existed in several versions since its conception in 1957. This report describes attempts to generalize one version, GPS-2-5; i.e., to have it solve many different kinds of problems. GPS's performance on eleven different tasks is discussed.

Several approaches to the construction of a general problem solver are surveyed in order to place GPS in perspective with other approaches and to formulate a meaningful problem of generality. A description of the organization and problem solving techniques of GPS is given, followed by a description of the representation of tasks.

The initial need for generalizing the representation of GPS-2-5 stems from the inadequacy of its representation for some tasks. However, the majority of the representational issues investigated in this research are concerned with the interaction between the problem solving techniques of GPS and its representation of tasks. The main consideration in generalizing the representation of GPS-2-5 is that problem solving techniques are applicable only if processes exist that can abstract certain information from the representation of a task.

TABLE OF CONTENTS

CHAPTER I: INTRODUCTION

The research reported here is an investigation into the development of a computer program with general problem solving capabilities. This investigation involved the construction of one such computer program called the General Problem Solver (GPS, although more properly GPS-2-6) which was accomplished by modifying an existing program called GPS-2-5. Both of these programs are derived from a computer program conceived in 1957 by A. Newell, J. C. Shaw, and H. A. Simon.

The emphasis in this research is on the generality of GPS--on the variety of problems which GPS can attempt to solve. The quality of the problem solving exhibited by GPS is only a secondary consideration. Hence, the kind of problems for which GPS was designed are simple according to human standards. A typical problem is the missionaries and cannibals task in which there are three missionaries and three cannibals who want to cross a river. The only means of conveyance is a small boat with a capacity of two people, which all six know how to operate. If, at any time, there are more cannibals than missionaries on either side of the river, those missionaries will be eaten by the cannibals. How can all six get across the river without any missionaries being eaten?

Another sample task is that of integrating, symbolically, a simple integral such as,

$$\int te^{t^2}\,dt.$$

This problem is apparently quite different from the missionaries and and cannibals task, but GPS has the generality, as well as the ability, to solve both of these problems.

Although GPS-2-5 was designed to be general, it, together with its predecessors, only solved three different kinds of problems due mainly

to inadequate facilities for representing tasks. The central problem of this research is to generalize GPS-2-5 so that it can attempt a wider variety of problems. We also demand that the formulation of problems for GPS requires no knowledge of the internal structure of the program. Underlying this specific objective is the desire to shed light on some of the issues involved in designing better representations for problem solvers.

This research does not endeavor to construct an impressive problem solver. Difficulties in reworking an existing program, such as GPS-2-5, make this infeasible. For example, the representation of tasks in GPS is somewhat ad hoc, having been introduced in several stages. (The representation of GPS-2-5 is a modification of the representation of a previous version of the program.) Thus, GPS is an experimental program used to investigate representational issues. No attempt to redesign the representation of GPS is discussed in this report.

This is a brief informal statement of the problem. Chapter II gives a more precise statement of the problem on which this research focuses. Chapter III describes the organization of GPS, which is essentially the same as that of GPS-2-5. The generalized representation of tasks is described in Chapter IV, while considerations in generalizing GPS-2-5 are discussed in Chapter V. The formulations for GPS of eleven different tasks are given in Chapter VI, together with the behavior exhibited by GPS in attempting to solve these tasks.

The generalization of GPS focused on the properties of a group of tasks. These tasks were singled out for reasons that are neither arbitrary nor entirely justified. Some of these tasks were successfully solved by GPS while others could not be solved by GPS. We shall return to this issue

at the end of Chapter V. Several of the tasks were deliberately selected because they have been solved by other problem solvers. The reason for giving such tasks to GPS is not to compare its performance with the performance of other problem solvers. Indeed, in all such cases, GPS is the more inefficient. However, giving these tasks to GPS is instructive because it helps to reveal the structure of the tasks, and the differences and similarities between GPS and other problem solvers.

GPS is programmed in IPL-V (Newell, et al [33]), a list processing language. This document does not require the reader to have an intimate knowledge of IPL-V. However, the reader should understand the concept of list processing.

CHAPTER II: THE ISSUE OF GENERALITY

This chapter states more precisely the problem of generality and the goals of this research. We start by illustrating the various approaches of current research to the construction of a general problem solver. With this background we formulate a version of the "problem of generality" that allows us to outline this research. Finally, we provide some appropriate historical background.

A. APPROACHES TO GENERALITY

How might one go about creating a general problem solver? Consider the simple model of a problem solver in Fig. 1. The problem is initially expressed in some external representation, which is converted by a translator into an internal representation--an encoding of the external representation inside the computer. The internal representation is processed by a set of problem solving techniques, and the result of this processing is (hopefully) the solution.

According to this simple view, generality can be limited by the generality of any of the three parts: the external representation, the internal representation, or the collection of techniques. Although eventually all three parts must be dealt with, an approach can start by emphasizing a single one: To adopt a general external representation that is similar to the way problems occur in the real world; to adopt a general internal representation so that all problems can be homogeneously represented inside of the computer; or to develop a set of problem solving methods of universal applicability. Each of these approaches focuses on one of the three parts of the model of a problem solver in Fig. 1 and considers the other two to be subordinate. To clarify these different approaches, a discussion of each follows which includes some examples of

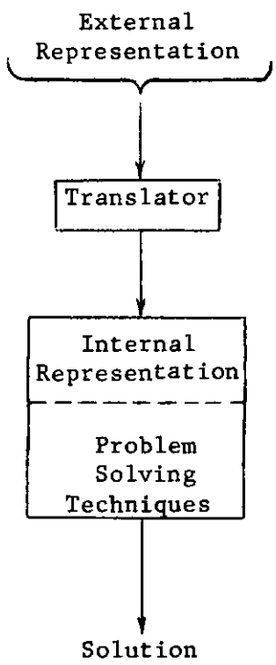


FIGURE 1. A simplified prototype of a problem solver.

research efforts that use the approach. This research adopts a variant (heuristic search) of the third approach. We will discuss it in great detail--the discussion of the other approaches serving to place matters in context.

External Representation

Two quite different possibilities seem to be appropriate for a "natural" representation of problems.

Natural Language Input. Assuming that problems are posed to the computer by people, a natural language is a good choice for an external representation of problems. Prior to the actual problem solving, the meaning must be extracted from sentences (and paragraphs) and encoded in the internal representation. The earlier work on question answering programs (SAD SAM in Lindsay [24] and BASEBALL in Green, et al [18]) focussed on this problem and purposely avoided any problem solving on the internal representation (although they did do sophisticated information retrieval.) Recently, however, several programs have endeavored to do problem solving on the meaning extracted from natural language inputs. The input language has had to be a much cruder approximation to a natural language than in the earlier question answering programs. A brief discussion of two of these gives some flavor of their structure.

1. SIR. SIR (Semantic Information Retriever, Raphael [49]) is a program which answers questions about a data base which it has accumulated from interaction with a human. A typical protocol is shown in Fig. 2.a. SIR answers the first two questions in Fig. 2.a, YES, because it knows that set-inclusion is both transitive and reflexive.

SIR extracts the semantics of statements and questions by matching each to a number of standard sentence forms. Associated with each form is a LISP (McCarthy, et al [29]) routine which stores information in the

(a) MAN: EVERY KEYPUNCH-OPERATOR IS A GIRL

SIR: I UNDERSTAND

MAN: ANY GIRL IS AN EXAMPLE OF A PERSON

SIR: I UNDERSTAND

MAN: IS A KEYPUNCH-OPERATOR A PERSON Q

SIR: YES

MAN: IS A PERSON A PERSON Q

SIR: YES

MAN: IS A PERSON A GIRL Q

SIR: SOMETIMES

MAN: IS A MONKEY A KEYPUNCH-OPERATOR Q

SIR: INSUFFICIENT INFORMATION

(b)

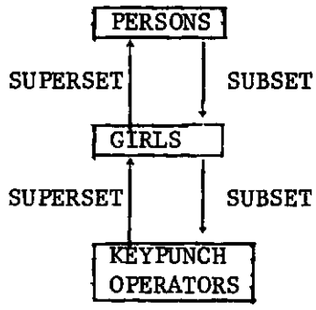


FIGURE 2. (b) is the SIR model of the conversation in (a) between SIR and a human.

internal represenation of the environment, or retrieves information from it as the case may be.

SIR's internal representation is a network in which the nodes represent nouns of English sentences and the branches between the nodes represent relationships between nouns. For example, the SIR model of the conversation in Fig. 2.a is given in Fig. 2.b. The nodes of the network represent GIRLS, PERSONS, and KEYPUNCH OPERATORS. The branch between GIRLS and PERSONS, which is labeled SUBSET, represents the fact, the set of all girls is contained in the set of all persons.

2. STUDENT. A program called STUDENT (Bobrow [5]) attempts to solve story algebra problems found in a high-school algebra textbook. While problems are posed to both STUDENT and SIR in a restricted subset of English, STUDENT unlike SIR is strongly oriented to a particular type of problem. Fig. 3.a is a typical question which was posed to STUDENT, together with STUDENT's answer.

The internal representation of problems in STUDENT is a set of algebraic equations. For example, Fig. 3.b is the set of equations which STUDENT arrives at for the problem in Fig. 3.a. Before solving the equation, STUDENT must recognize that the pairs of phrases in Fig. 3.c represent the same entity.

The main emphasis in STUDENT is translating the external representation. Since the internal representation is strongly task oriented, an algorithm can be used to solve the problem from its internal representation.

Visual Perception. The other "natural" external representation for many problems is the world itself. This external representation places large constraints on the problem solver. For example, to percieve the world it must be capable of accepting parallel inputs. In addition, answers must be produced quickly in order to solve problems in real time. Models

(a) THE GAS CONSUMPTION OF MY CAR IS 15 MILES PER GALLON . THE DISTANCE BETWEEN BOSTON AND NEW YORK IS 250 MILES . WHAT IS THE NUMBER OF GALLONS OF GAS USED ON A TRIP BETWEEN NEW YORK AND BOSTON Q

(b) 1. (EQUAL (DISTANCE BETWEEN BOSTON AND NEW YORK) (TIMES (250 (MILES)))
2. (EQUAL X1 (NUMBER OF GALLONS OF GAS USED ON TRIP BETWEEN NEW YORK AND BOSTON)
3. (EQUAL (GAS CONSUMPTION OF MY CAR) (QUOTIENT (TIMES 15 (MILES)) (TIMES 1 (GALLONS))))
4. (EQUAL (DISTANCE) TIMES (GAS CONSUMPTION) (NUMBER OF GALLONS OF GAS USED)))

(c) 1. GAS CONSUMPTION = GAS CONSUMPTION OF MY CAR
2. DISTANCE = DISTANCE BETWEEN BOSTON AND NEW YORK
3. NUMBER OF GALLONS OF GAS USED = NUMBER OF GALLONS OF GAS USED ON A TRIP BETWEEN NEW YORK AND BOSTON

FIGURE 3. (a) is a typical problem for STUDENT. (b) is the set of equations derived from (a). (c) is assumptions made by STUDENT in order to solve (a).

of pseudo-neural nets, e.g., perceptrons (Rosenblatt [41]) accept parallel inputs and produce answers in real time.

The basic element of pseudo-neural nets is, usually, an "adaptive threshold element" illustrated in Fig. 4.a. Each of the stimulus signals, $s_1, s_2, \ldots s_n$ may have either 1 or 0 as a value. The response signal, r, is determined by the sum of the stimulus signals times their corresponding weights, $w_1, w_2, \ldots w_n$. If

$$\sum_{i=1}^{n} w_i s_i > w_{n+1},$$

then r is 1; otherwise r is 0. A pseudo-neural net is a number of interconnected adaptive threshold elements such as the one illustrated in Fig. 4.b.

A problem for such a net takes the form of discriminating between two sets of stimulus patterns--those for which the correct response is 1 and those for which the correct response is 0. If the net has a large enough capacity there is likely to exist a set of weights, such that, for every input stimulus, it will produce the correct response. To find such a set of weights, stimulus patterns are presented to the neural net and depending upon the correctness of the response, the weights are adjusted so as to reinforce the net either positively or negatively.

It has been proven that, if a net has the capability of discriminating between two sets of stimulus patterns, certain reinforcement rules will eventually lead to a correct assignment of weights. However, the training sequence required to arrive at a correct assignment of weights may be arbitrarily long.

Adaptive pseudo-neural nets can be viewed as the type of problem solver depicted in Fig. 1. The external representation of a problem is the

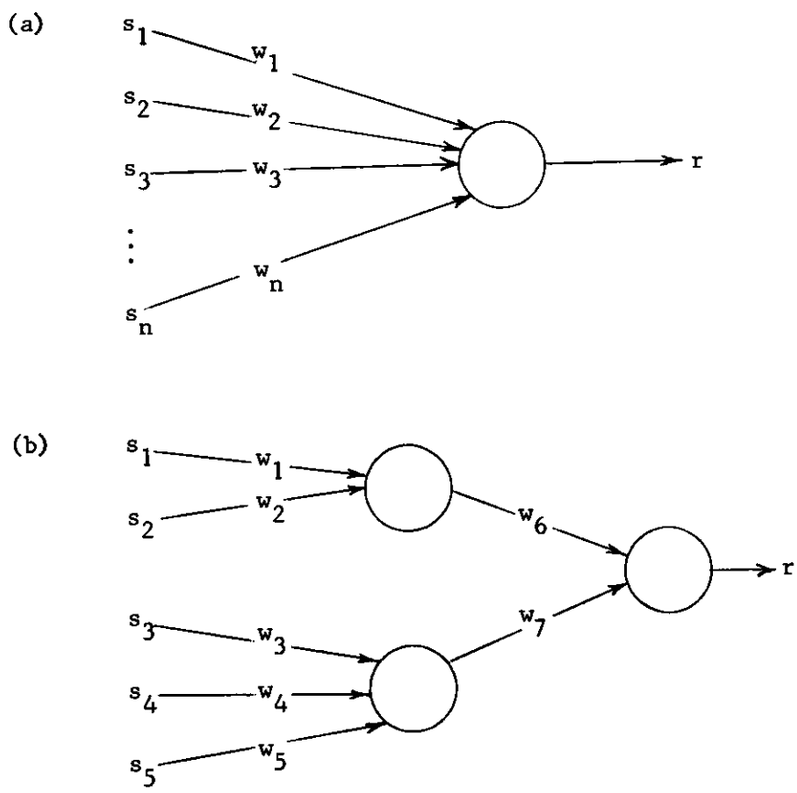


FIGURE 4. (a) is a threshold element and (b) is a simple pseudo-neural net.

two sets of stimulus patterns--those for which the correct response is 0 and those for which the correct response is 1. The internal representation of the problem is the values assigned to the weights of the adaptive threshold elements, since the values of these weights determine the response for any particular stimulus. In general different weights assignments cause a net to discriminate between different sets of stimulus patterns. The single problem solving technique employed by the adaptive net is the reinforcement of weights. Each reinforcement of the weights changes the internal representation of the problem and hopefully after a sufficiently large number of reinforcements the weights will have values that cause the net to perform the correct discrimination.

Internal Representation

In focusing on internal representation, we look for one that permits many different problems to be expressed in it. In addition, the internal representation should have a simple formal structure so that problem solving techniques that process the internal representation can be programmed. The first order predicate calculus is a general, formal system for expressing problems and, as such, is a good candidate for the internal represenation of a general problem solver. In many cases, the formulation of problems in the predicate calculus is somewhat clumsy. But the structure of this calculus is specific enough so that several programs which attempt to prove theorems expressed in the predicate calculus have been implemented. (Davis and Putnam [9]; Friedman [14]; Gilmore [17]; Robinson [50,51]; Wang [62,63]; Wos [67].

It is known that no mechanical proof procedure for the first order predicate calculus can guarantee an answer in a finite amount of time. However, the proof procedures have undergone successive reductions and purifications so that by now they have a simple and definite structure.

The internal representation of some recent theorem provers such as in Robinson [50], and their proof process is described on pages 201-214, since GPS used this formulation in proving a theorem in the predicate calculus.

The predicate calculus can conveniently be viewed as an internal representation because it does not contain information peculiar to any particular task. For example, the tasks for the theorem prover described in Robinson [50] are taken from group theory and number theory and the predicate calculus contains no information peculiar to either of these mathematical theories. Similarly, the proof process contains no task dependent information. The theorem prover in Robinson [50], contains a single rule of inference that combines two predicate calculus statements to form a new statement. A theorem is proved by assuming the negation of the theorem and inferring a contradiction.

Most of the work in theorem proving programs in the predicate calculus has focussed on the problem of attaining proofs. However, there has been some effort, identified mainly with work on the Advice Taker (McCarthy [26]) to extend the domain of problems for which the predicate calculus can be used as a representation. This work has focussed on problems of everyday reasoning and so far has been limited to toy problems. An example is a program by Black [4]. Fig. 5 is a simple problem which might be posed to Black's program.

A problem statement consists of a "corpus", which is a set of "unconditional statements" and "conditional statements" (statements which contain a '$\rightarrow$'), and a "question" which is an unconditional statement. The external representation of Black's program is very similar to its internal representation and the translation from one to the other is simple.

(a)

English	Internal Representation
CORPUS:	
I am at the desk.	AT(I, DESK)
The desk is at home.	AT(DESK, HOME)
If x is at y and y is at z, then x is at z.	AT(X, Y), AT(Y, Z)→AT(X, Z)
QUESTION:	
Am I at home?	AT(I, HOME)

(b)

English	Internal Representation
If desk is at the home, I am at home.	AT(DESK, HOME)→AT(I, HOME)

FIGURE 5. (a) is a simple problem for Black's program, and (b) is a "conditional statement" which is deduced by the program in finding a solution to (a).

Black's program can deduce a new statement from a "conditional statement" and an "unconditional statement." For example, to solve the problem in Fig. 5.a, the program would substitute I for X and HOME for Z in the third statement. From the first and third statement, it would deduce the statement in Fig. 5.b. The problem would be solved by deducing the question from the statement in Fig. 5.b and the second statement in Fig. 5.a.

Problem Solving Techniques

As the third alternative, we can focus on the techniques used to solve problems, ignoring temporarily both what internal representation will be used and the translation of the original problem into it. There may be many highly particular techniques; we will consider this case first. Alternatively, the techniques may be small in number with wide applicability. We consider an example of this, heuristic search, which forms the basis of the approach of this research.

Programming Languages. The development of problem oriented programming languages has continually eased the task of giving the computer a program for computing the answer to a problem. Constructing programs can be viewed as selecting a highly specific problem solving technique by combining the problem solving methods incorporated in the programming languages, e.g., the iteration statement in ALGOL. One can thus view the development of programming languages, as the creation of more powerful problem solvers.

The Turing Machine (i.e., a system defined as in Davis [8]) is capable of solving all problems that can be solved in a finite amount of time; however, its problem solving methods are extremely fundamental, and describing problem solving techniques to a Turing Machine is extremely laborious. To describe a problem solving technique to a modern general purpose computer is considerably easier than describing it to a Turing Machine. The problem solving techniques built into the former

(multiplication, random access memory, etc.) are considerably more elaborate than those which a Turing Machine possesses.

The problem solving techniques built into a problem oriented language such as ALGOL are considerably more powerful than the unit actions of a general purpose computer. Some powerful problem solving techniques can be easily specified by an ALGOL program. Finally, the problem solving techniques of ALGOL can be supplemented by a library of standard ALGOL subroutines, such as analysis of variance routines, linear programming routines, etc., which themselves are quite general problem solving techniques. Such a system has powerful problem solving capabilities and many new problem solving techniques can be specified easily.

ALGOL programs are usually not considered problem solving techniques. One reason is that most ALGOL programs are deterministic algorithms, whereas most problem solving programs, such as game playing programs. are heuristic programs that may or may not produce a solution to the problem. However, this distinction between deterministic and non-deterministic programs is more a reflection of the nature of the problem than the approach to solving the problem. Since ALGOL is designed for numerical problems, it is difficult to see the relationship between ALGOL and problem solving techniques required for complex non-numerical problems such as playing checkers.

To illustrate more cogently the relation of a programming language to problem solving techniques, consider the programming language, GPL (Game Playing Language in Williams [66][1]) that was designed for expressing procedures for playing board games and card games. The specifications of a game in GPL consists of

---

[1]An interpreter for GPL has been implemented in IPL-V.

a. data structures that describe the objects used in playing the game;

b. a procedure for playing the game.

For example, the specification of tic-tac-toe consists of a description of a tic-tac-toe board which is initially empty and the following procedure, expressed in GPL, for playing the game:

1 If there is a winning move then make it
else

2 If there is a winning move for the opponent, then block the opponent by making the move else

3 Make any legal move.

This procedure contains a definition of the legal moves of tic-tac-toe as well as a simple strategy for playing the game.

GPL was designed so that games can be described in it as briefly as they are described in a book of Hoyle, such as Morehead and Mott-Smith [31]. Strategy statements like "make a winning move" can also be specified briefly in GPL because it has the ability to search a board for a particular pattern as a primitive operation of the language. In the tic-tac-toe example, a winning pattern is a rank, file, or diagonal that has X's (or 0's as the case may be) on two squares and nothing on the third square. Thus, the primitives of GPL are general problem solving techniques for card and board games and they can be readily combined to form a specialized problem solving technique for a particular game.

DEDUCOM (Slagle [60] is another work which confounds the distinction between programming and problem solving. In many respects, DEDUCOM is similar to Black's program (discussed on pages 13-15). In part, the specification of a problem for DEDUCOM is a group of linguistic expressions that are combined to form new expressions during problem solving. However,

the specification of a problem may also contain LISP (McCarthy, et al [29]) expressions (which are programs) freely intermixed with other expressions. When they occur as subexpressions within a linguistic expression, they are executed by the standard LISP interpreter. When they contain linguistic expressions as subparts, interpretation is held up until the linguistic expressions can be solved. Consequently the problem solving power of DUDECOM stems in part from the ability of the LISP interpreter; and also from the freedom never to distinguish whether one is programming or writing down a problem.

Heuristic Search. A final way to focus on the generality of a problem solving technique is first to find a paradigm of a problem and then develop methods which are applicable to the paradigm. The generality of the paradigm determines the generality of the methods which are applicable to the paradigm. The paradigm need not imply a uniform representation of problems, but only that all problems which fit the paradigm have some common structure.

One general paradigm, which we shall call heuristic search (Newell and Ernst [38]), consists of two basic kinds of entities -- operators and objects. An operator, when applied to an object, produces a new object or indicates inapplicability. A heuristic search problem is:

Given
a. an initial situation represented as an object;
b. a desired situation represented as an object;
c. a set of operators.

Find a sequence of operators that will transform the initial situation into the desired situation.

The first operator of the solution sequence is applied to the initial situation, the other operators are applied to the result of the application of the preceding operator, and the result of the application of the last operator in the sequence is the desired situation.

The operators are rules for generating objects and thus define a tree of objects. Each node of the tree represents an object, and each

branch of a node represents the application of an operator to the object represented by the node. The node to which a branch leads represents the object produced by the application of the operator. In Fig. 6 for example, node A1 represents the object, A1, and branch X5 from A1 represents the application of the operator X5 to A1 which produces the object, A5.

A method for solving a heuristic search problem is searching the tree defined by the initial situation and the operators for a path from the initial situation to the desired situation. For example, if a problem has A0 as the initial situation and X1, X2,...as operators, the problem can be solved by searching the tree in Fig. 6 for a path from the top node of the tree to the desired situation. If the problem's desired situation is A5, a solution is (X1, X5). (Others might exist.)

An operator can, in general, only be applied to certain objects; it is infeasible to apply it to others. Consider the following example from arithmetic. It is infeasible to apply the operator,

$$x + y = y + x, \tag{1}$$

in which x and y are variables, to the object,

$$2 * 5.$$

In the extreme case each operator of a task would only be applicable to a single object. For example, the commutativity of addition could be represented as the list of expressions,

$$1 + 2 = 2 + 1 \tag{2}$$
$$1 + 3 = 3 + 1 \tag{3}$$
etc.,

assuming that numbers are bounded. Each expression such as (2) and (3) is considered a separate operator even though they all perform the same

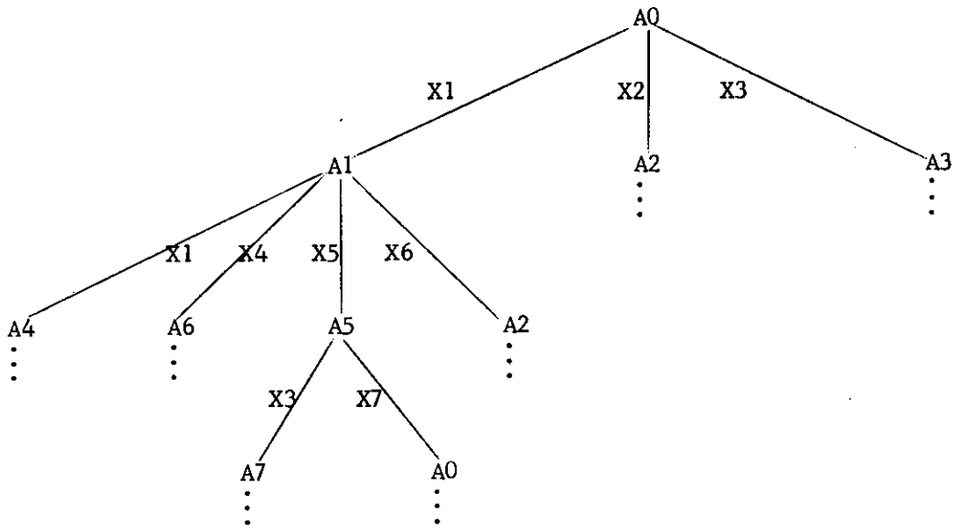


FIGURE 6. A typical object tree defined by a simple heuristic search problem.

function. In general, an operator is any function whose domain and range are objects, provided that the function is represenced as a single entity (e.g., a single routine, a single data structure, etc.). The reason for this unusual definition of operators is that problem solvers process single entities. For example, in applying the commutativity of addition represented as the list, (2), (3), etc., a problem solver must match the input object to the left side of many different operators in order to find the feasible member. On the other hand, when the commutativity of addition is represented as (1), the input object need only be matched to

$$x + y.$$

The effectiveness of a heuristic search problem solver is determined by its rules for selecting operators to be tried (rules for guiding the search). There are two basic criteria for selecting operators:

a. Desirability--the operator should produce an object which is similar to the desired situation;

b. Feasibility--the operator should be applicable to its input object.

The problem solver must face the dilemma that, in general, only one of these criteria can be satisfied, i.e., operators which are seemingly desirable are infeasible.

Some problem solvers which use heuristic search insist on the perfect desirability of operators, i.e., only those operators which produce the object desired are applied. Such problem solvers do not search the tree defined by the initial situation and the operator for the desired situation, but, rather, search the tree defined by the inverse operators and the desired situation, for the initial situation. (Q' is the inverse of the operator Q, if Q(A) = A' implies that Q' (A') = A for all A and

A'.) In this case, the problem solver works "backwards" and the desirability of the "forward problem" is nothing more than the feasibility of the backward problem. In general, there can be no gain in such a formulation because the backward, backward problem is the original forward problem. But for many problems working in one direction is considerably easier than working in the other direction, i.e., the space to be searched is considerably smaller.

Many different problems can be formulated in the heuristic search model of a problem, and this paradigm provides the underlying conceptual framework of many problem solving programs. All of the early game playing programs and theorem proving programs use the heuristic search model. We give below brief discussions of several efforts to give a flavor of how widespread the paradigm is. To do this we need to generalize the simplified heuristic search paradigm described above. (Schema, in the following, is an expression containing variables. An instance of the schema can be obtained by the substitution of constants for variables.)

a. The initial situation may be more than one object which can be represented by an object schema or a list of objects and object schemas.

b. The desired situation may be more than one object. It can be represented by an object schema, a list of objects and object schemas, by a more complex description, or by a testing procedure which can recognize it.

c. The operators may be given by schemas.

d. Some of the operators may have several objects as an input or may produce several objects as output.

e. The solution to the problem may be more complex than the simple sequence of operators described above. For example, if an operator has several objects as an output, for each output object, a sequence of operators which transform it into the desired situation may be required. In this case, the solution is a tree of operators.

1. Chess. Playing chess is an example of a problem which can easily be cast into the heuristic search paradigm. The initial situation is the chess position at the point of play and the operators are the legal chess moves. The desired situation is a chess position in which the opponent is checkmated. The move to be made from the initial position is the first move in the solution sequence.

To find such a move involves exploring the tree of possibilities. That is, for any move explore the possibilities that are available to the opponent, and for each of these, explore the possible responses, etc. This tree is so large for chess that for most chess positions, it is not possible to find a forced checkmate. However, exploring the tree does reveal important features hidden in a chess position and is a powerful method for evaluating a chess position.

Several programs which play chess (Baylor [2]; Bernstein, et al [3]; Kister, et al [21]; Kotok [22]; Newell, et al [41]) have been constructed. All of them use as their basic problem solving technique searching the tree defined by the initial chess position and the legal chess moves. These programs do not look for a checkmate but instead look for good positions. The goodness of a chess position is determined by an evaluation function which is designed to rate a position according to standard features, e.g., material advantage, mobility, center control, etc. Some programs have more elaborate evaluation functions than others and some use heuristic rules to guide the search. But all of the programs play chess by generating possible chess positions and using an evaluating function to determine the goodness of the position.

2. Other Games. In addition to chess programs, there exist several other game playing programs which use heuristic search; one plays checkers

(Samuel [53]); one plays Kahla (McCarthy [28]); one plays three dimensional tic-tac-toe (Gilbert [16]); and one plays five-in-a-row (Weizenbaum [64]). The framework of all of these programs is quite similar to that of the chess programs. The game positions are the objects and the legal moves are the operators. The initial situation is the initial game position and the desired situation is a class of objects. The desired situation is, in general, too remote and the program looks for a good move instead. A small part of the tree defined by the initial situation and the operators is searched and the best move is determined by using an evaluation function and a minimax procedure.

3. Propositional Calculus. LT (Logic Theory Machine in Newell, et al [39]) proves theorems in the sentential calculus of Whitehead and Russell. The initial situation is a set of objects each of which represents an axiom or a previously proven theorem. Each object is a group of primitive propositions combined according to the logical connectives, negation ($\sim$), conjunction ($\vee$), disjunction ($\wedge$), and implication ($\supset$). The desired situation is an object which represents the theorem to be proven. The operators are the rules of inference,

a. modus ponens--from A and $A \supset B$, B can be inferred

b. syllogism--from $A \supset B$ and $B \supset C$, $A \supset C$ can be inferred.

(A, B, C are variables for which propositions can be substituted.)

LT does not work forward but instead searches for a member of the initial situation in the tree defined by the desired situation and the inverse of the operators.

4. Geometry. Many of the theorems found in a high school Euclidian geometry textbook can be proven by the Geometry Machine (Gelernter [15]), a program which uses heuristic search. The initial situation is a set of objects each of which represents a premise of the theorem to be proven, e.g., angle ABC equals angle ABD. The conclusion of the theorem to be proven is the desired situation. Those theorems accepted as already proven are the operators. Theorems are proven by working backwards. Consider, for example, the desired situation that triangle ABC and triangle EFG are congruent. The inverse of the operator--if the three corresponding sides of two triangles are equal, then the triangles are congruent--could be applied to the desired situation producing three new objects,

a. segment AB equals segment EF
b. segment BC equals segment FG
c. segment CA equals segment GE

Each of these three objects must be inferred from the axioms and the premises of the theorem, in order for the theorem to be proven.

All the inverse operators have a single object as an input and one or more objects as an output. In this formulation a proof is not a simple path from the desired situation to the initial situation. Instead, it is a tree in which

a. the top node is the desired situation.
b. all of the terminal nodes are objects which are members of the set of objects representing the initial situation.
c. the immediate subnodes of a node represent the objects produced by the application of an inverse operator to the object represented by the node.

5. Integration. SAINT (Slagle [59]) is a computer program which integrates expressions symbolically. The initial situation is the expression to be integrated and the desired situation is a set of objects, each of which represents standard integral forms. The operators are "heuristic transformations" for changing the form of an object. One of the operators, for example, is the substitution of tan u for x in an object in which x is the variable of integration. If the initial situation were

$$\int \frac{dx}{1+x^2}$$

the application of this operator to it would result in the new object

$$\int du \ .$$

In applying an operator SAINT automatically performs "algorithm-like transformations" such as algebraic simplification and differentiation.

6. Programming. Amarel [1] developed a program which constructs programs in a highly task oriented programming language. The objects are flow diagram schemas--flow diagrams in which some of the actions might be variables. The operators are rules for flow diagram modification--substitution of a specific action or an "elementary" flow diagram schema for a variable action in a flow diagram schema. A typical sequence of objects generated by Amarel's program is illustrated in Fig. 7.

The program to be constructed is described by listing all of its inputs along with their corresponding outputs; thus, the desired situation is an object which, when executed on the inputs, produces the corresponding outputs. (There are special mechanisms to deal with the facts that some of the actions may be variables and that there may be a large number of input, output pairs.)

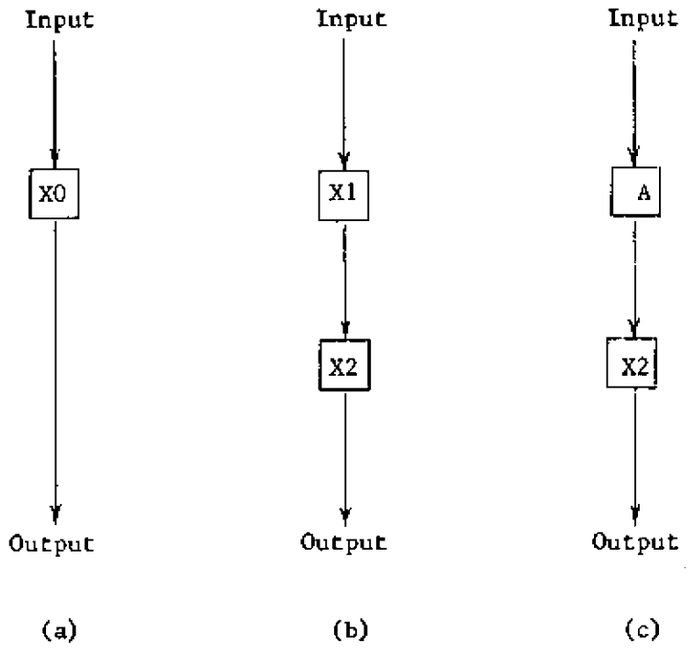


FIGURE 7. Typical flow diagram schemas in which A is a specific action and X0, X1, X2 are variable actions. (a) is the initial situation of the task. (b) is produced by substituting a flow diagram schema for X0 in (a). (c) is produced from (b) by substituting A for X1 in (b).

7. Everyday Reasoning. Even some of the research efforts focusing on internal representation use heuristic search as their basic problem solving method. Black's program (described on pages 13-15) treats "unconditional statements" as objects and "conditional statements" as operators. The desired situation is the "question" and the initial situation is the set of "unconditional statements" in the "corpus".

Black's program works "backwards". The inverse operators all have a single object for an input and produce one or more objects as a result of their application. For this reason, a solution is not a simple path from the desired situation to the initial situation but a tree which has the same form as a solution found by the Geometry Machine.

8. Predicate Calculus. The proof process of some contemporary programs which prove theorems in the first order predicate calculus (e.g., the one described on pages 12-13) can be viewed as heuristic search. The objects are statements and the single rule of inference is the only operator. The initial situation is the set of statements whose conjunction is the negation of the theorem to be proven. The desired situation is a contradictory statement.

The operator has two objects as an input and produces a single object as a result. The input objects to the operator must be in the set of objects representing the initial situation or be the result of a previous application or the operator.

9. Story Algebra Problems. STUDENT's only problem solving technique is an algorithm for solving a set of simultaneous equations symbolically (see page 8). Thus, STUDENT does not use heuristic search. However, the problem of solving a set of simultaneous equations can be easily formulated as a heuristic search problem by treating the equations as objects

and algebraic manipulations as operators.[2]

GPS. All of the programs discussed above use heuristic search because it is a convenient framework for the particular problem and not because it is a general paradigm for solving problems. On the other hand, GPS is an attempt to implement problem solving techniques that have general applicability to heuristic search problems. GPS uses the heuristic search paradigm directly; a problem is given to GPS in terms of objects and operators.

GPS attempts problems by tree search, as in any heuristic search program. But GPS employs a general technique called means-ends analysis to guide the search, which involves subdividing a problem into easier subproblems. Means-ends analysis is accomplished by taking differences between what is given and what is wanted, e.g., between two objects or between an object and the class of objects to which an operator can be applied. A difference designates some feature of an object which is incorrect. GPS uses the difference to select a desirable operator--one which is relevant to reducing the difference. For example, in attempting the original problem, GPS detects a difference, if one exists, between the initial situation and the desired situation. Assuming that a desirable operator exists and that it can be applied to the initial situation, GPS applies the operator to the initial situation which results in a new object. GPS rephrases the original problem by replacing the initial situation with the new object and then recycles. As usual, the problem is

---

[2]This formulation is used in Krulee and Kuck [23].

solved when an object is generated which is identical to the desired situation.

If an operator is not applicable to an object, an attempt to apply it will result in a difference--the reason it is not applicable. If the difference is not too difficult, GPS will attempt to alleviate the difference in the same way that it attempts to reduce a difference between two objects. If the attempt to reduce the difference is successful, a new object will be produced and, hopefully, the operator can be applied to the new object.

Previous versions of GPS have solved several tasks. The first task was a formulation of proving theorems in the propositional calculus designed by Moore [30]. Fig. 8.a shows a typical problem which GPS solved. In addition to the problem in Fig. 8.a, GPS had to be given the information in Fig. 8.b (called the task environment). The DIFF-ORDERING[3] orders the differences of this task according to their relative difficulty; the TABLE-OF-CONNECTIONS associates with each difference those operators which are relevant to reducing it. The objects and operators are represented by schemas. Fig. 8.c illustrates the internal representation of an object and Fig. 8.d illustrates the internal representation of an operator.

Another task solved by a previous version of GPS is the missionaries and cannibals task (described in Chapter I). The objects were configurations of people on the river banks and were represented by schemas. The operators which moved people across the river could not be completely

---

3Words written with all capital letters correspond directly to IPL symbols used in the IPL routines that comprise GPS.

(a) Initial Situation: $(R \supset -P) \cdot (-R \supset Q)$

Desired Situation: $-(-Q \cdot P)$

(b) Operators:

R1 $AVB \rightarrow BVA$, $A \cdot B \rightarrow B \cdot A$

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

R3 $AVA \rightarrow A$, $A \cdot A \rightarrow A$

R4 $AV(BVC) \leftrightarrow (AVB)VC$, $A \cdot (B \cdot C) \leftrightarrow (A.B) \cdot C$

R5 $AVB \leftrightarrow -(-A \cdot -B)$

R6 $A \supset B \leftrightarrow -AVB$

R7 $AV(B \cdot C) \leftrightarrow (AVB)\ (AVC)$, $A \cdot (BVC) \leftrightarrow (A \cdot B)V(A \cdot C)$

R8 $A.B \rightarrow A$, $A.B \rightarrow B$

R9 $A \rightarrow AVX$ (X is any expression)

R10 $[A, B] \rightarrow A.B$ (Two expressions input)

R11 $[A \supset B, A] \rightarrow B$ (Two expressions input)

R12 $[A \supset B, B \supset C] \rightarrow A \supset C$ (Two expressions input)

DIFF-ORDERING

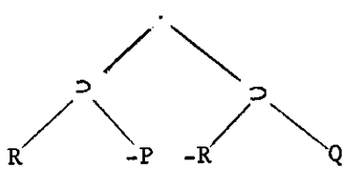
Add Variables, Decrease Variables
Increase Number of Variables, Decrease Number of Variables
Change Connective
Change Sign
Change Grouping
Change Position

FIGURE 8. (continued on next page)

TABLE-OF-CONNECTIONS:

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12
Add Variables									X	X		X
Decrease Variables								X			X	X
Increase Number of Variables			X				X		X	X		X
Decrease Number of Variables			X				X				X	X
Change Connective					X	X	X					
Change Sign		X			X	X						
Change Grouping				X			X					
Change Position	X	X										

(c)



(d)

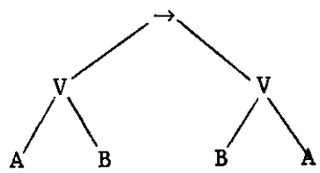


FIGURE 8. (a) is a problem in the propositional calculus which GPS solved. (b) is information which must be given to GPS in addition to the problem. (c) is the internal representation of an object and (d) is the internal represnetation of an operator.

represented by schemas; instead the operators were represented by expressions together with a special routine which interpreted the semantics of the expression. For example, the expression

L: MC

represents the operator which moves one missionary, one cannibal, and the boat from the right bank to the left bank. Given an object and this expression the special routine created a new object in which one missionary, one cannibal and the boat were moved across the river. The new object would only be produced if the boat was originally on the right bank and if no missionaries could get eaten, etc. One disadvantage of using the special routine is that its construction requires knowledge of the internal structure of GPS.

The only other task given to GPS previous to this report is a task found in a psychological experiment designed to investigate the mathematical ability of children (approximately age 7). The problem is to transform one string of I's and O's into another by

a. adding two I's to the right end of the string;
b. adding two O's to the right end of the string;
c. deleting one I;
d. deleting one O.

GPS solved the problem of transforming

IOIIOIOOI

into

IOOIOOOII.

The objects are the strings of I's and O's which were represented by schemas. The operators are the rules for modifying the strings and were

also represented by schemas.

Several tasks have been formulated for GPS but never carried through to completion. Proving trigometric identities was proposed and hand simulated (Newell, et al [42]). The problem of balancing an assembly line (the task solved by another heuristic program) was formulated in terms of GPS (Tonge [61]).

A formulation for GPS of the problem of discovering a good set of differences for a task was proposed in Newell, et al [45].

Two programs have been constructed which have deliberately adapted the problem solving techniques of GPS to a particular task.

1. Heuristic Compiler. The Heuristic Compiler (Simon [54]) is a program which constructs programs in IPL-V (Newell [33]). It consists of three basic parts each of which corresponds to a GPS "task environment" (a type of task for GPS). A description of one, the "State Description Compiler", is sufficient to illustrate how the problem solving techniques of GPS can be applied to a programming task.

In the state description of a programming task, the initial situation is the list of cells affected by the execution of the program together with the contents of each cell prior to execution. The desired situation is the contents of the affected cells after the execution of the program. For example, Fig. 9.a represents the programming task of replacing the contents of the SIGNAL-CELL by MINUS. (IPL-V is a list processing language and PUSHDOWN1 represents the second symbol on the list, SIGN-CELL.) In Fig. 9.b PUSHDOWN1 and SYMB1 represent an arbitrary symbol. An operator is a previously compiled routine and is represented by a list of the cells affected by its execution together with the contents of these cells before and after its execution, e.g., the operator

(a)	Affected Cells	SIGNAL-CELL	
	Input	SYMB1, PUSHDOWN1	
	Output	MINUS, PUSHDOWN1	
(b)	Affected Cells	ACCUMULATOR	CELL1
	Input	SYMB2, PUSHDOWN1	SYMB1, PUSHDOWN2
	Output	PUSHDOWN1	SYMB2, PUSHDOWN2
(c)	Affected Cells	ACCUMULATOR	
	Input	PUSHDOWN1	
	Output	SYMB1	

FIGURE 9. (a) is the state description of a programming task. (b) and (c) are state descriptions of two previously compiled routines.

shown in Fig. 9.b.

In attempting to construct the program represented in Fig. 9.a, the Heuristic Compiler notices the difference that the contents of the SIGNAL-CELL is not MINUS. According to its TABLE-OF-CONNECTIONS, the operator shown in Fig. 9.b is relevant to reducing this difference. After substituting MINUS for SYMB2, so that the operator will perform a desirable function, the Heuristic Compiler notices that the operator cannot be applied because of the difference that the contents of the ACCUMULATOR is not MINUS. This difference is reduced by applying the operator in Fig. 9.c after substituting MINUS for SYMB1. The operator in Fig. 9.b can be applied to the resulting object and the task is solved.

2. Binary Choice Experiment. In the binary choice, the subject is asked to predict which of two events will occur in each of a series of trials. The subject is told which event occurs after he makes his prediction. A program which uses GPS's problem solving methods was constructed to simulate human behavior in the binary choice experiment (Feldman, et al [12]). The assumption underlying this simulation is that the subject is entertaining hypotheses about the pattern of events which have occurred.

The objects of this task are hypotheses about the pattern of events and the operators are rules for forming hypotheses according to the actual sequence of events. The differences are features of hypotheses, which, according to the actual sequence of events, seem to be incorrect. Thus, GPS's methods are not used to predict events but to form a model of the environment which is used to predict events.

B. POSING A PROBLEM OF GENERALITY

As we have seen, there are several quite distinct ways to approach constructing a problem solver with some degree of generality. Their diversity underscores that the decision to work with GPS entails selecting a particular approach: one that derives its appeal from the wide applicability of heuristic search, but that ignores by and large the way problems are represented externally or internally. The importance of the internal representation will become evident in what follows. Nevertheless, the internal representation used in GPS was chosen ad hoc, within the framework of the problem solving techniques to be used, and not as the primary consideration in implementing GPS.

It is clear that generality has to do with the size of the domain of problems that can be handled by a problem solver. Still it is not enough to specify just the problem domain in evaluating the generality of a program. It is the purpose of this section to clarify some of the additional considerations so that we can finally state a meaningful problem of generality.

Amount of Specification

If we were to take seriously that generality is defined by the domain of problems which are solvable, then there are many perfectly general problem solvers: Turing Machines, ALGOL compilers, etc. But the generality of a Turing Machine (for example) stems from the fact that the amount of information in the specification of a problem to a Turing Machine is not limited. For example, the problem of playing perfect chess could be represented by all possible chess positions together with the best move for each position. It is theoretically possible to give this information to a Turing Machine, since the number

of different chess positions is finite. But this specification, being impractical, does not qualify a Turing Machine as a chess player.

The advantage that ALGOL has over a Turing Machine (or an assembly language) is that most problems of interest can be specified more briefly in ALGOL. Consider the problem of evaluating a polynomial. To describe this problem to a Turing Machine, it would be necessary to also describe numerical operations (multiplication, addition, etc.). The problem can be specified in ALGOL by a single declarative statement. The information in the specification of a problem determines whether the problem solving is done endogenously or exogenously. In describing the evaluation of a polynomial to a Turning Machine most of the problem solving techniques were contained in the specification of the problem. On the other hand, an ALGOL translator has built into it problem solving techniques which are sufficient to evaluate polynomials.

Thus, the generality of a problem solver must be defined relative to the amount of information it takes to specify a problem. An ALGOL translator would appear to be more general than an assembly language or a Turing Machine because in ALGOL problems can be described in terms of more sophisticated concepts such as iteration statements.

Problems are specified in terms of the concepts built into the problem solver. In constructing a general problem solver, we face the dilemma that the concepts built into it should be both sophisticated and general. The sophistication of the concepts allows the problem specification to be brief while the generality of the concepts allows them to be useful in specifying more than one problem. Chess programs, for example, contain the concept of playing chess and the problem of finding a move for a particular chess position is specified by specifying the

chess position. Although the concept of playing chess is a very sophisticated concept, it is also very specialized. The concept of playing a game on a chess board is a more general concept; both chess and checkers could be specified in terms of this concept. However, the specification of chess in terms of a game on a chess board, would necessarily include the definition of the legal chess moves as well as the chess position.

We know of no way to determine, for any particular task, what information should be built into the problem solver and what information should be contained in the specification of the task. But clearly this issue is relevant to the evaluation of a general problem solver.

Quality of Problem Solving

An outstanding property of the various efforts to construct a general problem solver is that the quality of the problem solving suffers as generality of the problem solver is increased. For example, the best chess program (Kotok [22]) in existence plays a modest[4] game of chess. Although GPS can attempt more than one kind of problem, the only kind of problems that it can solve are considerably easier than chess. The representation in GPS of the chess board and the legal chess moves would be cumbersome and GPS's problem solving techniques are not sufficient to play even poor chess.

The power of a problem solver is determined by the effectiveness of its problem solving techniques while its generality is determined by

[4]It has trouble checkmating a beginner while it will put up a fight against a good chess player.

the domain of problems to which the techniques are applicable. Each technique requires that certain information be abstracted from the internal representation. The techniques are applicable if processes can be found which abstract the necessary information from the internal representation. For example, one of the requirements of the techniques of GPS is that in attempting to apply an operator to an object, a new object is produced if the operator is applicable to the object; otherwise, a difference is produced. Thus, there must be a process which, given an operator and an object expressed in internal representation, will either produce a difference or an object expressed in the internal representation, depending on the applicability of the operator to the object.

The internal representation is pulled in two directions: on the one hand, it must be general so that problems can be translated into it, and, on the other hand, it must be specific enough for the problem solving techniques to be applicable. Thus, there are many different generality problems, one for each set of problem solving techniques and the difficulty of a particular generality problem depends on the variety and complexity of the techniques. If this were not the case, a problem solver, more general than any in existence, could be constructed by using a natural language for its internal representation and giving it no problem solving techniques. Of course, it would never solve a problem, regardless how trivial, but it would be very general.

More cogently, it would be much easier to achieve generality with a problem solver that only did forward search by applying the operators in a fixed order than with GPS. Conversely, it would be more difficult to achieve the level of generality that we have achieved for a problem

solver that is more adequate than GPS.

Role of Representation

A problem is expressed quite differently for different problem solvers. For example, a story algebra problem is expressed in English for STUDENT. A story algebra problem can also be expressed in the first order predicate calculus or as a heuristic search problem in terms of objects and operators. Although each of these formulations represents the same problem in some sense and thus are isomorphic to each other, are they really the same problem? A human presented with these formulations would probably exhibit considerably different behavior in finding a solution for each and would probably not recognize they really are the same problem.

Most problems can be formulated several different ways, and each formulation will demand different kinds of processing. Contrast the problem of integrating an expression using only elementary integral forms to the problem of integrating the same expression using an integral table. Both problems are isomorphic and, in fact, the forms in the integral table can be derived from the elementary forms assuming the knowledge of certain trigometric identities and algebraic manipulations. However, a problem solver using the integral table must be capable of processing large amounts of data while the problem solver which uses only the elementary forms must be considerably more clever than the other problem solver.

Perhaps it is best to consider different formulations of a problem to be different problems. Unfortunately, this raises other questions on comparing the performance of problem solvers that have different internal representations.

Summary

A meaningful problem of this research can finally be formulated--to extend the generality of GPS while holding its power at a fixed level. This involves extending the internal representation of GPS in such a way that its problem solving methods remain applicable and in a way that increases the domain of problems that can be translated into its internal representation. Thus, this research is mainly concerned with representational issues. We would not expect the issues to be the same in generalizing the internal representation of a problem solver which employed different techniques than GPS. In this respect, this research has the nature of a case study.

Two representational issues were discussed in this section:

a. the amount of information that is required to specify a problem;

b. which of several isomorphic representations is a neutral representation of a problem.

These issues, although important, are only secondary concerns of this research. The primary concerns are to discover the way in which the problem solving techniques interact with the internal representation, and to learn something about the properties of a good internal representation for the problem solving techniques of GPS.

Let us recapitulate the plan of the research now that the task is clear. Chapter III describes the problem solving techniques of GPS and Chapter IV the generalized internal representation. We keep these quite distinct so that we can essentially hold the techniques constant while modifying the internal representation to meet the demands of generality. In Chapter V, the interaction between the internal representation and the techniques is illustrated by examining the nature of modifications necess-

ary to get GPS to work on different tasks. Chapter VI describes the different tasks actually given to GPS; these illustrate the generality of GPS as well as its power. Finally, Chapter VII provides a summary.

C. HISTORY OF GPS

Since this report is concerned intensively with GPS, a brief description of the different versions of GPS is appropriate. (The following is all of the published material either describing GPS or discussing its use in the simulation of cognitive processes: Newell [34,35,36]; Newell, et al [40,42,45]; Newell and Simon [43,44]; Simon and Newell [56,57,58]). GPS grew out of the Logic Theory Machine (described on pages 24-25), a program for proving theorems in the sentential calculus of Whitehead and Russell. The first version, called GPS-1 was coded in IPL-IV for JOHNNIAC, a Princeton class computer at the RAND Corporation. All of the other versions have been coded in IPL-V (Newell [33]). The successor of GPS-1, called GPS-2-1, was similar to GPS-1, functionally, but organizationally was quite different. The change to GPS-2-2, the next version of GPS, involved smaller organizational changes but required a separate designation since, for a short period, both versions were operational. This version is rather completely documented (Newell [35]).

GPS-2-3 changed the internal represenation. Objects and operators were now represented by description lists--attribute-value pairs--instead of by conventional lists which were used in previous versions. GPS-2-4 was obtained by revising the mechanism for testing the identity of two data structures. In the predecessors of GPS-2-4, there were several ad hoc processes for testing if two data structures of a particular type are identical, e.g., two goals or two objects. In GPS-2-4 these ad hoc

processes were replaced by a general process for testing the identity of any two data structures regardless if they are goals, objects, or whatever.[5]

GPS-2-5 introduced a language for describing problem solving methods that allowed the application of a method to be monitored by the problem solving executive. Thus it incorporated both a major change in internal representation and in problem solving organization over GPS-2-2.

The version of the program used in this research started with GPS-2-5. The problem solving structure was not altered but the internal representation was generalized under the impact of new tasks. Although this current version should be called GPS-2-6, for expediency it is called simply GPS.

All the IPL-V versions of GPS (GPS-2-1 to GPS-2-5) were run on the IBM 7090. The current version has been run on the CDC G21, a machine with 65K of 32-bit memory (requiring two words per IPL symbol).

GPS was produced by five successive modifications of GPS-2-1 over a five-year period. Some of the programming conventions have become confusing and a significant portion of the code is ad hoc. This makes description more difficult and muddies somewhat the lessons to be drawn from generalizing GPS-2-5. In fact, there now seems little further profit in continuing with this version rather than constructing an entirely new GPS program.

---

[5]This process is described on pages 57-61.

A certain degree of success was guaranteed because the previous versions of GPS had moderate problem solving capabilities. On the other hand, serious programming difficulties had already been encountered and modification could be expected to introduce more. Consequently, no high expectations were held for the power of the problem solving to be shown by GPS across many tasks.

One serious limitation on the expected performance of GPS is the size of the program and the size of its rather elaborate data structure. The program itself occupies a significant portion of the computer memory and the generation of new data structures during problem solving quickly exhausts the remaining memory. Thus, GPS is only designed to solve modest problems whose representation is not too elaborate. Although larger computers' memories would alleviate the extravagances of GPS's use of memory, conceptual difficulties would still remain. For example, GPS never erases any goals or objects generated during problem solving.

CHAPTER III: THE PROBLEM SOLVING STRUCTURE OF GPS

The simple scheme of Fig. 1 may be used to show the overall organization of GPS. This chapter describes the problem solving techniques of GPS. The details of the internal representation are ignored in this chapter. We assume there is some encoding of objects, operators, and differences which the problem solving techniques can process. The internal and external representations of a task are described in the next chapter.

The problem solving techniques are organized by GOALs[1]. That is, the main function of the problem solving techniques is to achieve GOALs and in the process other GOALs may be generated to which the problem solving techniques are also applied. GOALs which are discussed in the first section of this chapter, are achieved by applying relevant methods[2]. The methods are expressed in a special method language, which is described in the second section. The PROBLEM-SOLVING-EXECUTIVE, described in the third section of this chapter, selects and applies methods. In the last section, each method is described individually.

A. GOALS

A GOAL is a data structure which consists of the information that defines a desired state of affairs plus a history of previous attempts to achieve the GOAL. A GOAL provides sufficient context for problem solving activity. That is, in any context GPS can stop what it is doing and start working on a new GOAL or on a previous GOAL, where it left off.

---

[1] As previously noted, words written with all capital letters have a direct correspondence to IPL symbols in GPS.

[2] We have used the word "techniques" rather than "method" in the preceding, since the methods in GPS have a highly precise definition.

The statement of a problem must be formulated as a GPS GOAL. GPS uses only four types of GOALs. (The necessity for others has not arisen.) The four types of GOALs are:

a. TRANSFORM object A into object B. To achieve this GOAL a series of objects, which are derived[3] from A, is generated. The final member of the series is identical to B.

b. REDUCE difference D on object A. To achieve this GOAL GPS produces a new object A', which is derived from A. The feature of A to which D refers is modified in A'.

c. APPLY operator Q to object A. To achieve this GOAL a new object is generated by applying Q to A or some object derived from A.

d. SELECT the elements of set S which best fulfill criterion C. To achieve this GOAL an element of S is selected. C is stated with respect to an object, e.g., select the element of S most similar to object A.

A typical example of how GPS subdivides GOALs into simpler GOALs is represented by the tree of GOALs in Fig. 10. The original GOAL is G1. In attempting to achieve this GOAL, GPS notices the difference, D, between A and B and creates G2. GPS attempts to modify A by creating G3. The operator Q is not applicable to A, but the difference D' is detected and G4 is created. (Note that D' is related to applying Q, not to the original GOAL, G1.) G4 is achieved by the solution of G5. GPS then continues working on G3 by creating G6, which uses the result of G4, A'. The successful application of Q to A' results in the solution of G3 and G2, both of which use the result of G6, A'', as their results. In reattempting the original GOAL, G7 is created and G1 will be achieved,

[3]Object A is derived from object B if it is produced by the application of an operator to B or some other object derived from B.

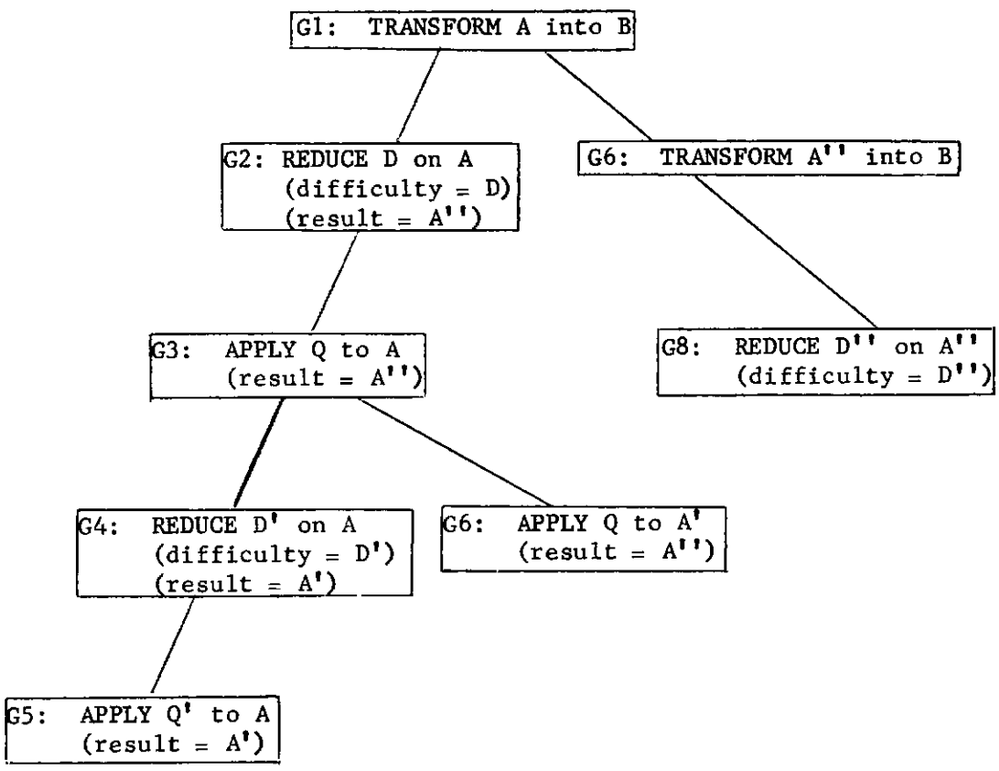


FIGURE 10. A tpyical GOAL tree showing the difficulty and the results of the GOALs.

if G7 is successful. However, the difference D'' between A'' and B is detected, and the process continues.

If GPS finds a sufficiently undesirable situation while attempting G7, the previous GOALs may be retried in hope of finding new results. But, the basic strategy of GPS is to continue on the current approach, rather than to do an exhaustive search for results.

B. METHOD LANGUAGE

The methods for achieving GOALs are expressed in a method language, and the PROBLEM-SOLVING-EXECUTIVE is an interpreter for this language. A system of signals provides the main means of communication between the methods and the PROBLEM-SOLVING-EXECUTIVE. A signal is a single IPL symbol. Each method, when executed, assigns to the variable, CURRENT-SIGNAL[4], a signal which summarizes the result of the method's execution. The next action of the PROBLEM-SOLVING-EXECUTIVE, following the execution of a method, depends on the value of the CURRENT-SIGNAL.

In the method language there are four different method structures, as far as interpretation is concerned. Three of these correspond to the primitive term, the unconditional expression, and the conditional expression, which are incorporated in almost all programming languages. The fourth, GOAL construction, is somewhat peculiar to problem solving.

An IPL method, which is the primitive term in the method language, is an IPL subroutine. This type of method is executed by calling the IPL interpreter to execute it. No further methods occur inside an IPL method.

4CURRENT-SIGNAL is the name of the IPL cell in which the methods put the signal that summarizes their execution. However, it may be more convenient to view CURRENT-SIGNAL as a variable which is assigned a value during the execution of a method.

A SEQUENTIAL method is a list of methods used for unconditional operation. Such a method is executed by executing, in the order of their occurrence, all of the methods on the list. The execution of the sequence may be terminated part way through, but otherwise is unconditional. Both in a SEQUENTIAL method and a SIGNAL-LIST method (described next) any of the four types of methods can occur; thus full phrase structure is permitted.

A SIGNAL-LIST method is used for conditional operation. It consists of a list of pairs, each of which is a signal followed by a method. A SIGNAL-LIST method is executed by executing the sub-method which is paired with the same signal as the CURRENT-SIGNAL. If none of the signals associated with the sub-methods is identical to the CURRENT-SIGNAL, then no sub-method is executed. Thus, this type of method performs an arbitrary n-way branch conditional on the CURRENT-SIGNAL.

A GOAL-SCHEMA method is a request for the construction of a GOAL. This type of method has the form of the GOAL to be constructed, stated relative to the CURRENT-GOAL context. CURRENT-GOAL is a variable whose value is the GOAL that GPS is currently attempting. An example of a GOAL-SCHEMA method is

> TRANSFORM the result of the last subgoal into
> the second object of the CURRENT-GOAL.

The result of the last subgoal and the second object of the CURRENT-GOAL depend on the context of the CURRENT-GOAL and, thus, a GOAL constructed according to this GOAL-SCHEMA depends on the CURRENT-GOAL context. The PROBLEM-SOLVING-EXECUTIVE executes a GOAL-SCHEMA method by constructing the GOAL, evaluating it and attempting it, if acceptable.

A SIGNAL-LIST method can be used to perform iterations by recursive

execution of its submethods. However, iterations can be performed more directly by having the PROBLEM-SOLVING-EXECUTIVE repeatedly execute a method as long as a certain condition is satisfied. For example, if a method is marked to repeat on SUCCESS, it will be repeated as long as the CURRENT-SIGNAL is a signal that indicates SUCCESS. Several different signals indicate the different degrees of SUCCESS while several other signals indicate the different kinds of FAILURE. Any type of method except a GOAL-SCHEMA method can be designated as repeatable. The conditions on which repetition can occur are:

a. a change in the CURRENT-SIGNAL;
b. the CURRENT-SIGNAL indicates FAILURE;
c. the CURRENT-SIGNAL indicates SUCCESS;
d. the CURRENT-SIGNAL does not indicate either SUCCESS or FAILURE.

Normally, whenever the CURRENT-SIGNAL indicates SUCCESS or FAILURE, the execution of a method is terminated. However, any non-repeatable method, except a GOAL-SCHEMA method, can be marked to continue on SUCCESS or FAILURE.

An example of a method is the GENERATE-AND-TEST-METHOD, shown in Fig. 11, which is used to achieve a SELECT GOAL. Fig. 11 and all of the other figures that define methods are placed at the end of the chapter to keep them together. In these figures the sub-methods followed by '(IPL)' are IPL methods. The 'sub-method' column of a SIGNAL-LIST method is divided into two parts: one or several signals appear in the left part; and a method appears in the right part. The method gets executed whenever the CURRENT-SIGNAL is any of the signals. For instance, in IS-IT-OK, BEGIN and TEST-PASSED both lead to executing the IPL method FIND-NEXT-TEST. A signal can occur as a sub-method of a SEQUENTIAL or SIGNAL-LIST method.

It is processed as if it were the IPL method which assigns the signal to be the value of the CURRENT-SIGNAL. Unless otherwise noted, the SEQUENTIAL and SIGNAL-LIST sub-methods that occur in the main method are also defined in the figure.

The GENERATE-AND-TEST-METHOD generates the elements of the SET one at a time and applies to each a series of tests. The first element which passes all of them is the element which is selected and marks the termination of the method.

The equivalent of the GENERATE-AND-TEST-METHOD is given in a flow chart in Fig. 12 for comparison. SELECT-MEMBERS may result in FAILURE which terminates the method and RECORD-RESULT will never be executed. As long as IS-IT-OK fails, select members will be repeated. However, if FIND-NEXT-MEMBER-OF-SET fails, SELECT-MEMBERS will be terminated because this type of FAILURE is UNCONDITIONAL-FAILURE. UNCONDITIONAL-FAILURE will terminate any method, overriding instructions to repeat or continue on FAILURE. IS-IT-OK will be repeated until a test fails or until all of the tests are passed. If the next test cannot be found, FAILURE is not reported. But this condition terminates the method because the CURRENT-SIGNAL will not change after this condition arises. Both FIND-NEXT-MEMBER-OF-SET and FIND-NEXT-TEST find the first as well as the next.

C. PROBLEM SOLVING EXECUTIVE

In addition to interpreting methods, the PROBLEM-SOLVING-EXECUTIVE performs the following functions:

a. METHOD-SELECTION
b. GOAL-RECOGNITION
c. GOAL-EVALUATION
d. OBJECT-RECOGNITION
e. OBJECT-EVALUATION

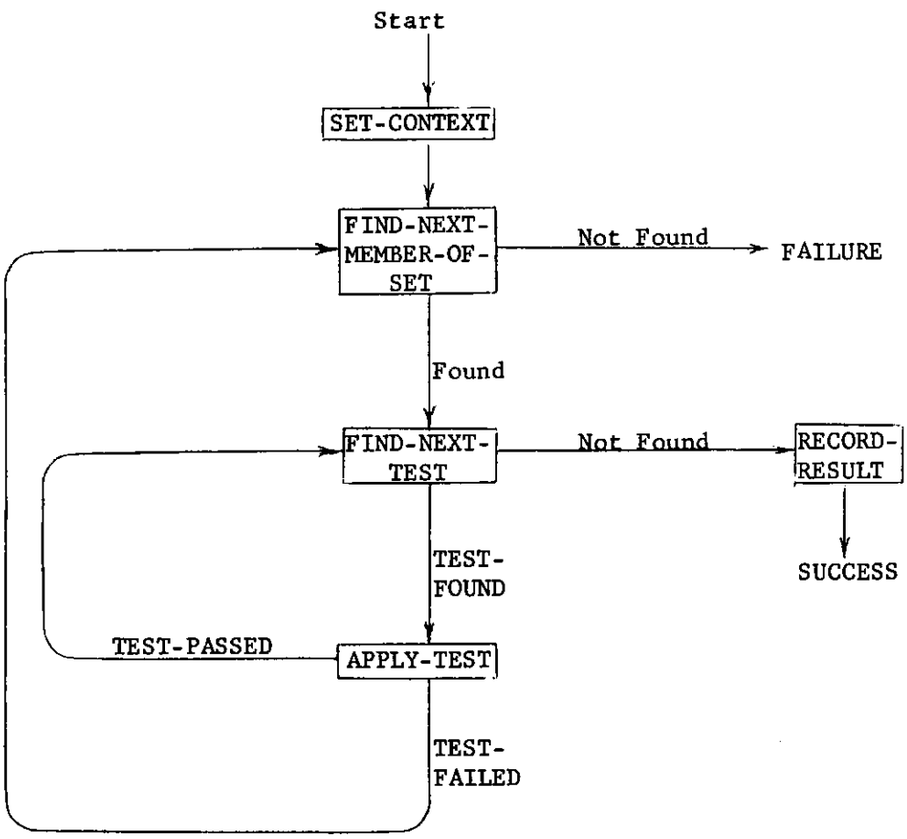


FIGURE 12. The flow chart representation of the GENERATE-AND-TEST-METHOD (compare with Fig. 11).

f. GOAL-SELECTION

g. RECORD-ATTEMPTS

h. SET-GOAL-CONTEXT

Each of these functions is a box in Fig. 13, a flow chart of the PROBLEM-SOLVING-EXECUTIVE, and is discussed individually later in this section. All of the other boxes in Fig. 13 pertain to the interpretation of the method language.

The PROBLEM-SOLVING-EXECUTIVE is always in the context of a GOAL. Initially it is in the context of the TOP-GOAL which is the statement of the problem, given in the specification of a task.

The PROBLEM-SOLVING-EXECUTIVE starts off by trying to select a method (see Fig. 13). If one is selected, it is attempted by first discriminating on the type of the method (i.e., their grammatical type). SEQUENTIAL and SIGNAL-LIST methods are attempted by trying their sub-methods one at a time. Thus, this processing is purely interpretive. On the other hand, the PROBLEM-SOLVING-EXECUTIVE constructs a GOAL in order to attempt a GOAL-SCHEMA method. After constructing the GOAL, the PROBLEM-SOLVING-EXECUTIVE files it (this process will be described in detail in GOAL-RECOGNITION) and recognizes if it is equivalent to a GOAL filed previously. If the GOAL passes an evaluation, the executive abandons the CURRENT-GOAL, after recording its status, and initializes the context for the new GOAL. The new GOAL is then attempted by selecting a method that is relevant to achieving it.

The executive uses the IPL interpreter to execute an IPL method. An IPL method might select an old GOAL which the executive will evaluate to decide if it should be attempted. IPL methods can also produce objects, which are then evaluated by the executive. An undesirable object will cause the executive to abandon the GOAL. If the object produced is a new

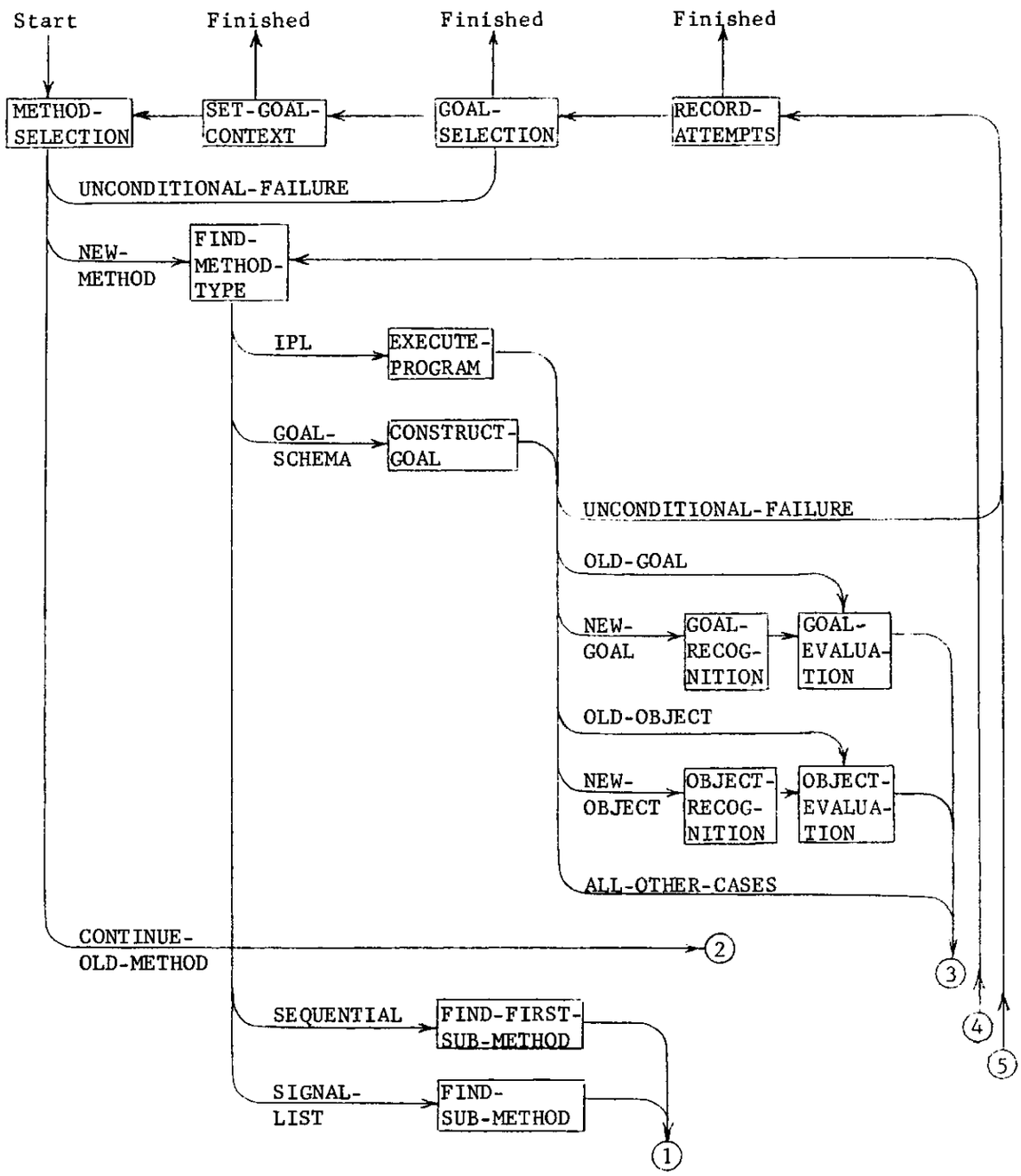


FIGURE 13. (continued on next page)

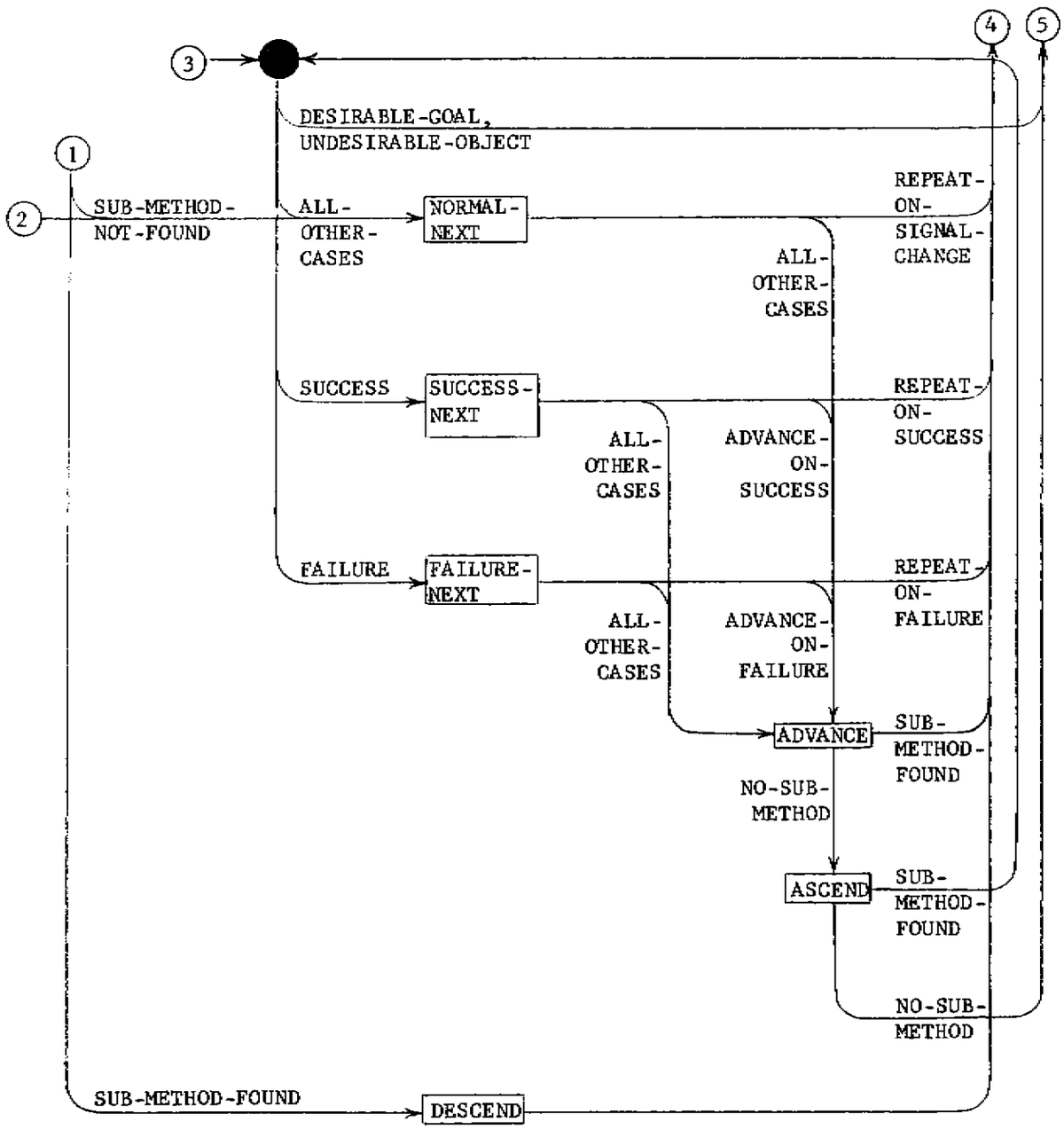


FIGURE 13. The flow-chart of the PROBLEM-SOLVING-EXECUTIVE.

one, the executive will file and recognize it in the same way that it recognizes GOALs, before evaluating it.

METHOD-SELECTION

METHOD-SELECTION is done by a discrimination tree, shown in Fig. 14. The terminal nodes of the tree are methods. The selection is performed by discriminating at nodes starting with the top node, and then at the node resulting from the previous discrimination, until arriving at a terminal node. The method at the terminal node is the one selected, provided that its status for the CURRENT-GOAL is not EXHAUSTED. If the discrimination at any node does not yield a new node, or if the method at the terminal node is EXHAUSTED, all methods are EXHAUSTED and the selection results in UNCONDITIONAL-FAILURE.

At every node the discrimination is on the feature of the current context, enclosed in the box representing the node. For example, if a method is being selected for a new GOAL, the next discrimination will be on the GOAL-TYPE of the CURRENT-GOAL. If it is a TRANSFORM GOAL whose given object is a SET of objects, the TRANSFORM-SET-METHOD will be selected.

Some of the discriminations depend on the representation of the task, e.g., TYPE-OF-OPERATOR. Such discrimination will be clarified in the next chapter, which discusses the details of the representation of a task.

GOAL-RECOGNITION

In GPS the philosophy for comparing two data structures that are not atomic symbols, such as GOALs, is to assign to them canonical names and compare only their names. The canonization is accomplished by a general process, similar to an EPAM net (Feigenbaum [10]) for filing data structures. Whenever a new data structure (one without a canonical

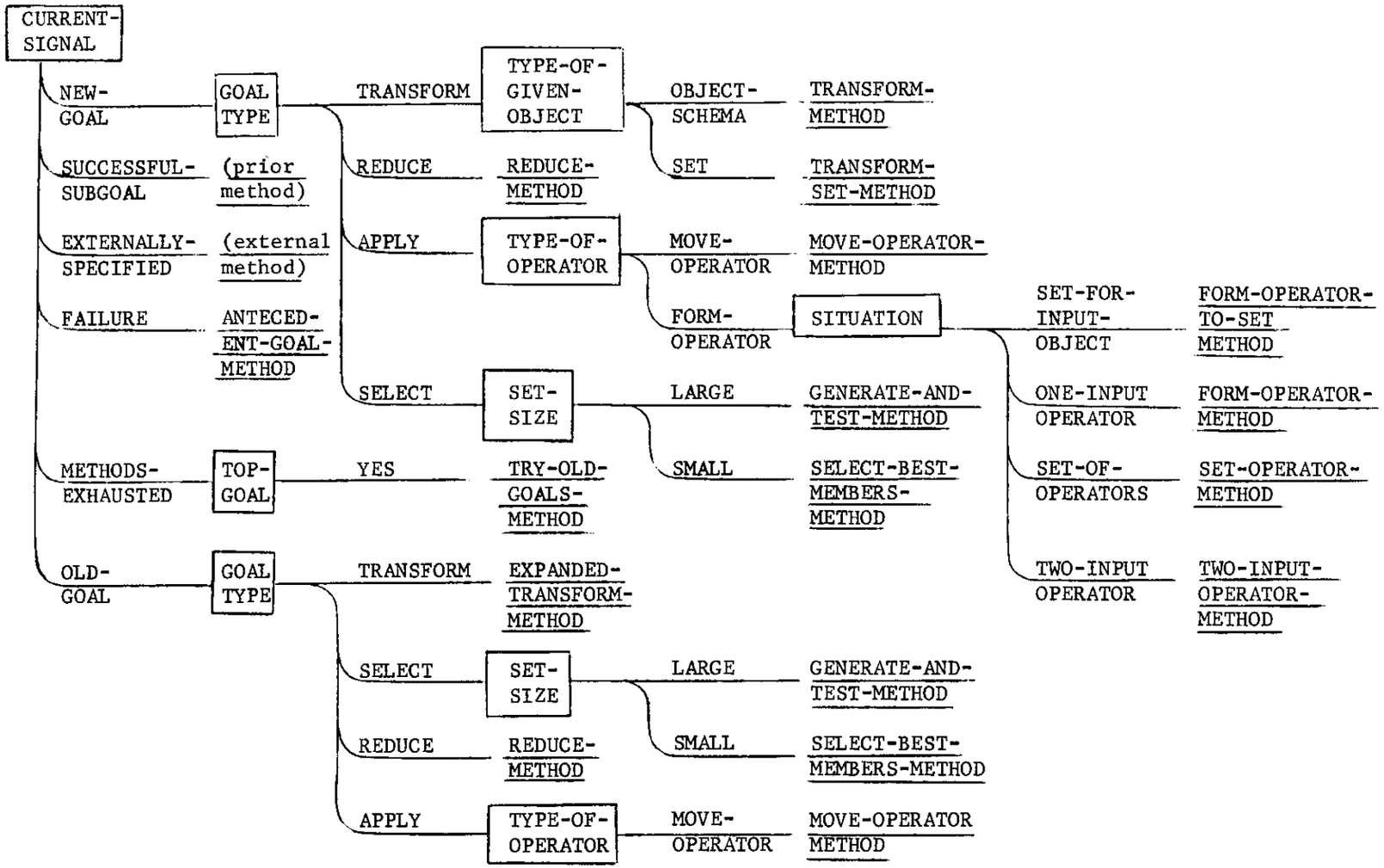


FIGURE 14. The discrimination net used for METHOD-SELECTION.

name) is encountered, it is filed. If the data structure is identical to one already filed, it is replaced by the filed structure which, by definition, is the canonical structure; otherwise, it is filed and becomes a canonical structure.

In GPS data structures are filed in a discrimination tree like the one shown in Fig. 15.a. The nodes of the tree, except for terminal nodes, are properties of data structures and the branches represent values of the properties. The terminal nodes represent filed data structures. Filing a data structure involves sorting it through the tree and, if necessary, growing the tree so that it can be filed at a terminal node.

Consider the example of filing in the net in Fig. 15.a the GOAL

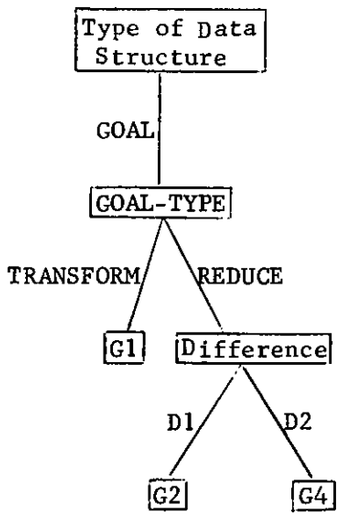
G5: TRANSFORM A1 into B1.

Discrimination on the property at the top node of the tree sorts G5 to the left most subnode of the top node because G5 is a GOAL. The discrimination on the GOAL-TYPE sorts G5 to the node, G1. Since G1 is a terminal node, the tree does not contain sufficient discrimination to distinguish between G1 and G5. G1 and G5 are matched and if they are identical, G1 is used as the canonical form of G5. (In this case, G5 is not filed.) On the other hand, if a difference is detected between G1 and G5, it is used to grow the tree so that G5 can be filed. The tree shown in Fig. 15.b is the result of filing G5 in the tree in Fig. 15.a, assuming that G1 is the GOAL,

G1: TRANSFORM A2 into B2.

There are two essential properties of this process. First, the data structure being filed will be matched to at most one other data

(a)



(b)

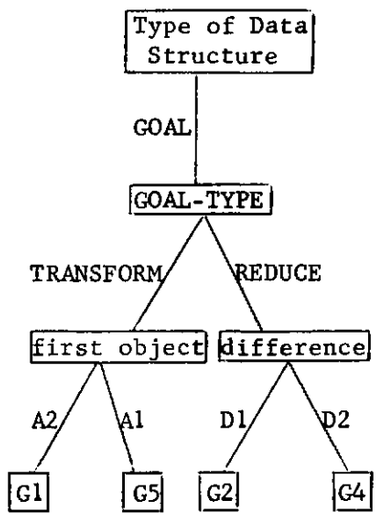


FIGURE 15. (b) is the result of filing G5 in the discrimination net, (a).

structure. The matching is done by the IDENTITY-MATCH-METHOD in Fig. 16 described later in this chapter. This keeps the amount of processing small, since matching is the most expensive part of the process when the data structure being filed is large. The other essential property of the filing process is its generality, i.e., the process can deal with other types of data structures in addition to GOALs. Both objects and differences are filed in the same tree.

If GOALs were not canonized, every GOAL (except the GOAL that is the statement of the problem) would have a unique supergoal. However, since GOALs are canonized, any GOAL may have several supergoals. To see this suppose that GOAL, G1 is created to simplify GOAL, G2 and that a previously generated GOAL, G3 is equivalent to G1. In canonizing G1, it is replaced by its equivalent, G3, and consequently, G3 has two supergoals: G2 and the original supergoal of G3. Thus, the canonization of GOALs causes the GOAL structure of GPS to be a GOAL network instead of a GOAL tree.

GOAL-EVALUATION

The basic strategy of GPS is that problem solving proceeds from difficult GOALs to easy GOALs. This strategy requires that none of the subgoals of a GOAL, G, are more difficult than G. Consequently, GPS considers any GOAL undesirable if its supergoal is easier than the GOAL.

The above strategy also requires the antecedent GOAL of a GOAL, G, to be no easier than G because an antecedent GOAL is the previous step in problem solving at the level of G. The antecedent GOAL of a GOAL, G, is defined as the GOAL whose result is used in the statement of G and whose supergoal is the same as the supergoal of G. (Currently, there is no need for more than a single antecedent.) In Fig. 10 for example, G2 is the antecedent GOAL of G7, and G4 is the antecedent GOAL of G6. No other GOAL

in Fig. 10 has an antecedent GOAL.

The difficulty of a GOAL is determined by the difficulty of the difference associated with the GOAL. Since REDUCE GOALs are the only type of GOALs which have differences associated with them, they are the only GOALs which can be evaluated according to their difficulty. However, other GOALs are considered as difficult as their most difficult subgoal.

For example, G7 in Fig. 10 does not have a difference associated with it. But GPS considers it as difficult as G8 because D'' is the most difficult difference detected in matching A'' and B; hence G8 is the most difficult subgoal of G7. GPS considers G8 desirable only if D'' is not more difficult than D because G2 is the antecedent GOAL of G7. Similarly, G4 is desirable only if D' is not more difficult than D because the supergoal of G3 is G2.

OBJECT-RECOGNITION

All newly generated objects are canonized in the same way that GOALs are canonized, using the same discrimination tree and filing processes. The main reason for canonizing objects is to simplify the canonization of GOALs. Since the names of objects appear in GOALs, matching two GOALs would necessitate matching objects which appear in the GOALs, if the canonical names of objects were not used.

OBJECT-EVALUATION

An object which is considerably larger than any of the objects in the TOP-GOAL, will be considered undesirable.

GOAL-SELECTION

Only simple GOAL selection is done directly by the problem solving executive; more complicated GOAL selection is accomplished by the

execution of a method whose purpose is to select a GOAL.[5]

If a new desirable GOAL, G, is generated, the PROBLEM-SOLVING-EXECUTIVE will abandon the CURRENT-GOAL and work on G. The supergoal is selected whenever a method for achieving a GOAL is completed (SUCCESS, NO-PROGRESS, or FAILURE). The PROBLEM-SOLVING-EXECUTIVE has no fixed search strategy, e.g., breadth first or depth first, built into it. Instead, GOALs are attempted iteratively by working on one GOAL until deciding to abandon it in order to work on another GOAL. However, if only these two rules were used to select GOALs, they would be attempted in the recursive order in which they are generated.

The only other GOAL selection rule employed directly by the PROBLEM-SOLVING-EXECUTIVE is that TOP-GOAL is selected whenever a newly generated GOAL is identical to a previously generated GOAL. This rule prevents GPS from entering an endless loop.

RECORD-ATTEMPTS

Before abandoning a GOAL, the PROBLEM-SOLVING-EXECUTIVE records certain information which summarizes the attempt to achieve the GOAL. For example, the methods that have been tried together with their status--EXHAUSTED or NOT-EXHAUSTED--are recorded.

SET-GOAL-CONTEXT

After selecting the next GOAL to be attempted, the context of this GOAL is initialized before selecting a method relevant to achieving the GOAL.

5The only methods whose purpose is to select GOALs are the TRY-OLD-GOALS-METHOD and ANTECEDENT-GOAL-METHOD.

D. METHODS

Each of GPS's methods, except for the GENERATE-AND-TEST-METHOD (described on pages 51-2) are described below. The figures that define the methods are at the end of the chapter. The details of some of the methods will not be entirely clear because they are dependent upon the representation of tasks, which is discussed in the next chapter.

TRANSFORM-METHOD

The TRANSFORM-METHOD is used for achieving the GOAL of transforming object, A, into object, B, and is defined in Fig. 17. After the context for the method is initialized, the two objects are matched by the MATCH-DIFF-METHOD which is defined in Fig. 18. If the two objects are not identical, MATCH-DIFF-METHOD detects the differences and the most difficult difference is determined by SELECT-DIFFERENCE. If no differences are found, SUCCESS is reported by REPORT-SUCCESS, which terminates the method.

On finding differences, the construction of a GOAL to reduce the most difficult difference on A is requested. If the GOAL fails, the method terminates with FAILURE; otherwise, the GOAL results in a new object, C, and the GOAL of transforming C into B is constructed. The result of this GOAL is used as the result of the CURRENT-GOAL.

MATCH-DIFF-METHOD

The MATCH-DIFF-METHOD, which is a sub-method of the TRANSFORM-METHOD, detects all of the differences between two data structures and is defined in Fig. 18. The strategy of this method is to divide data structures into parts and then match corresponding parts. On finding a difference, it tries to alleviate the difference with an immediate operator which is a simple transformation on objects. (Immediate operators are given as a

parameter to the method and are task dependent.) For example, if the type of difference--variable versus term--were detected, the immediate operator of substitution would be applied. The term would be substituted for the variable, and the difference would vanish.

The first parts of the two data structures are found by FIND-NEXT-PART, which uses the definition of parts that are given as a parameter to the method. Another parameter to the methods is the types of differences which should be detected. The parts are matched by checking for each type of difference, one at a time. When detected, a difference is reported, provided that the immediate operators cannot alleviate it. After checking for all the types of differences, the next parts are found and matched, etc. The method is finished when all of the parts are matched because CURRENT-SIGNAL does not change after this condition arises.

IDENTITY-MATCH-METHOD

The IDENTITY-MATCH-METHOD, defined in Fig. 16, is very similar to MATCH-DIFF-METHOD. In fact, the two sub-methods of the IDENTITY-MATCH-METHOD are also used in the MATCH-DIFF-METHOD. The IDENTITY-MATCH-METHOD is used by the canonization process to compare the identity of two data structures (see pages 57-61). It differs from the MATCH-DIFF-METHOD in two ways: It does not use immediate operators, and it terminates upon detecting a single difference, whereas the MATCH-DIFF-METHOD detects all of the differences.

REDUCE-METHOD

The REDUCE-METHOD, defined in Fig. 19, is the only method for achieving the GOAL of reducing a difference on an object. First, it selects a desirable operator by finding the next operator and testing its desirability. If the operator is undesirable, it finds the next operator,

etc. On finding a desirable operator, the GOAL of applying the operator to the object of the REDUCE GOAL is constructed which, if successful, will result in a new object. The new object is used as the result of the REDUCE GOAL.

Since there may be several desirable operators, all of which produce different objects, the GOAL may have several results. However, the method is terminated when an operator is successfully applied. The other results are found only by retrying the method.

FORM-OPERATOR-METHOD

In general, any operator can only be legally applied to certain objects. A FORM-OPERATOR can be applied to an object whose form is the same as the input form of the operator, and which satisfies the pretests of the operator.

The FORM-OPERATOR-METHOD defined in Fig. 20, is the method for achieving an APPLY GOAL whose operator is expressed as a FORM-OPERATOR. After the context is initialized, the applicability of the operator is tested by EXECUTE-PRETESTS and by matching the object to the input form. If no differences are detected, the operator is applicable, and its result which is used as the result of the CURRENT-GOAL is produced. If differences are detected, a GOAL to reduce the most difficult one is constructed.

If the REDUCE GOAL is successful, it results in a new object, and a GOAL to apply the operator to the new object is constructed. The result of the latter GOAL if it is successful, becomes the result of the CURRENT-GOAL. (Fig. 10 illustrates the way in which the result of a GOAL is also used as the result of its supergoal.)

FORM-OPERATOR-TO-SET-METHOD

The method for achieving the GOAL of applying a FORM-OPERATOR to a SET of objects is the FORM-OPERATOR-TO-SET-METHOD which is defined in Fig. 21. First, the object most similar to the input form of the operator is selected by a GOAL constructed for that purpose. Then, the GOAL of applying the operator to the object selected is generated and its result is also the result of the CURRENT-GOAL.

SET-OPERATOR-METHOD

The SET-OPERATOR-METHOD, defined in Fig. 22, is used to achieve an APPLY GOAL whose operator is a SET of FORM-OPERATORs. This method is the same as the FORM-OPERATOR-TO-SET-METHOD except that an operator, whose input form is similar to the object, is selected instead of the converse.

TWO-INPUT-OPERATOR-METHOD

Some form operators have two input forms instead of one. Such operators are applied to a pair of objects, both of which are derived from the same objects, e.g., both derived from the axioms of a theory. Each object has the same form as one of the input forms.[6]

The TWO-INPUT-OPERATOR-METHOD, defined in Fig. 23, is used to achieve an APPLY GOAL whose operator has two input forms. First, the operator is applied to the object of the CURRENT-GOAL by selecting one input form and applying it to the object. In applying the operator to the object, the operator is modified if it is necessary to substitute for some of the variables in the operator. The modified operator is the

6The logic operators, R10, R11, R12, in Fig. 8 are good examples of operators which have two objects as input.

result of applying the operator to the first input. The other input object is selected from the SET of objects that are derived from the same object that the first input object is derived from. After applying the operator to the second input, the new object is produced.

MOVE-OPERATOR-METHOD

The above methods for achieving APPLY GOALs all assume that the operator is represented as a FORM-OPERATOR. A MOVE-OPERATOR is an alternative representation for an operator. Unlike a FORM-OPERATOR, a MOVE-OPERATOR may produce several results, each of which is obtained from a different specification of the variables in the operator. The MOVE-OPERATOR-METHOD terminates on finding a single result, but may be retried.

Fig. 24 is the definition of the MOVE-OPERATOR-METHOD for achieving a GOAL whose operator is a MOVE-OPERATOR. After specifying the variables in the operator, the feasibility of the operator is tested. If feasible, i.e., no difference found, the resultant object is produced and tested for legality by executing the POST-TESTS. If the POST-TESTS fail, the resultant is rejected; otherwise, it is used as a result of the GOAL.

On the other hand, if a difference is found, a GOAL is set up to reduce the difference provided that it is not too difficult. If the REDUCE GOAL is successful, a GOAL is constructed to apply the operator to its result.

Upon retrying APPLY-FEASIBLE-OPERATOR a different specification of the variables is used, and the method may be retried as long as a new specification of variables is found.

TRANSFORM-SET-METHOD

The TRANSFORM-SET-METHOD, defined in Fig. 25, is used to achieve the GOAL of transforming a SET of objects into an object. It is very similar to the FORM-OPERATOR-TO-SET-METHOD in that the CURRENT-GOAL is rephrased by replacing the SET of objects by one of its members.

EXPANDED-TRANSFORM-METHOD

The EXPANDED-TRANSFORM-METHOD, defined in Fig. 26, is a method for achieving the TOP-GOAL. It replaces the initial situation of the TOP-GOAL by the SET of objects which were all derived from the initial situation. The rationale for this method is that the GOAL of transforming one of these objects into the desired situation might be easier than the TOP-GOAL. Since all of the objects are derived from the same objects, the solution of one GOAL is equivalent to the solution of another.

SELECT-BEST-MEMBERS-METHOD

Fig. 27 is the definition of the SELECT-BEST-MEMBERS-METHOD which is used to achieve a SELECT GOAL whose SET is "small". The SELECT-MEMBERS-METHOD applies a test to each member of the SET that passed all of the previous tests. If only one member passes the test, it is the member selected; if no members pass the test, those members which passed the previous test are selected; if more than one member passes the test, the procedure is repeated with the next test.

The SELECT-BEST-MEMBERS-METHOD applies a test to all members of the SET. Only those members that fail the test are eliminated. On the other hand, the GENERATE-AND-TEST-METHOD applies tests to one member at a time. Whenever a member is found that passes all of the tests, it is the member selected. However, there may be several other members which also would pass all of the tests and one of these members may fulfill the criterion

better than the one selected. The GENERATE-AND-TEST-METHOD is used for "large" SET because in most cases only several members of the SET need to be processed.

ANTECEDENT-GOAL-METHOD

The ANTECEDENT-GOAL-METHOD is used to achieve any GOAL that has an antecedent GOAL[7]. It is an IPL method that selects the antecedent GOAL to be retried. The rationale is that the antecedent GOAL might produce a new result and the CURRENT-GOAL can be rephrased in terms of the new result. In Fig. 10, for example, if G7 fails, its antecedent GOAL, G2, would be retried. If it produced the new result, A''', and if A''' were successfully transformed into B, G1 would be successful, which is the sole purpose of G7.

TRY-OLD-GOALS-METHOD

TRY-OLD-GOALS-METHOD is an IPL method which is the last resort in finding a solution. When all else fails, it selects a GOAL[8] provided that the methods for achieving it are not EXHAUSTED. REDUCE GOALs, whose supergoals are TRANSFORM GOALs, are retried before other unfinished GOALs.

7Antecedent GOAL is defined on pages 61-62.

8The difficulty of a GOAL is defined on pages 61-62.

Name and Kind of Method	Sub-Methods	
GENERATE-AND-TEST METHOD (SEQUENTIAL)	SET-CONTEXT(IPL)	
	SELECT-MEMBER	
	RECORD-RESULT (IPL)	
SELECT-MEMBER (SEQUENTIAL, repeat on FAILURE)	FIND-NEXT-MEMBER-OF-SET (IPL)	
	IS-IT-OK	
IS-IT-OK (SIGNAL-LIST, repeat until no signal change)	BEGIN, TEST-PASSED:	FIND-NEXT-TEST (IPL)
	TEST-FOUND:	APPLY-TEST (IPL)
	TEST-FAILED:	FAILURE

FIGURE 11. The definition of the GENERATE-AND-TEST-METHOD for achieving a SELECT GOAL whose SET is large.

Name and Kind of Method	Sub-Methods	
IDENTITY-MATCH-METHOD (SIGNAL-LIST, repeat on signal change)	BEGIN, PARTS-MATCHED, ONLY-ONE-PART-FOUND:	FIND-NEXT-PART (IPL)
	PARTS-FOUND, CONTINUE-MATCHING:	FIND-DIFFERENCE-BETWEEN-PARTS (IPL)

FIGURE 16. The definition of the IDENTITY-MATCH-METHOD which is used to test the identity of two data structures.

Name and Kind of Method	Sub-Methods
TRANSFORM-METHOD (SEQUENTIAL)	SET-CONTEXT (IPL)
	MATCH-DIFF-METHOD
	SELECT-DIFFERENCE (IPL)
	REPORT-SUCCESS
	REDUCE the difference selected on the first object of the CURRENT-GOAL (GOAL-SCHEMA)
	TRANSFORM the result of last subgoal into the second object of the CURRENT-GOAL (GOAL-SCHEMA)
	REPORT-RESULT (IPL)
REPORT-SUCCESS (SIGNAL-LIST)	NO-DIFFERENCES: SUCCESS

FIGURE 17. The definition of the TRANSFORM-METHOD used for achieving a TRANSFORM GOAL.

Name and Kind of Method	Sub-Methods	
MATCH-DIFF-METHOD (SIGNAL-LIST, repeat on signal change)	BEGIN, PARTS-MATCHED, ONLY-ONE-PART-FOUND:	FIND-NEXT-PART (IPL)
	PARTS-FOUND, CONTINUE-MATCHING:	FIND-DIFFERENCE-BETWEEN-PARTS (IPL)
	DIFFERENCE-FOUND:	PROCESS-DIFFERENCE
PROCESS-DIFFERENCE (SEQUENTIAL)	TRY-IMMEDIATE-OPERATORS (IPL)	
	RECORD	
RECORD (SIGNAL-LIST)	DIFFERENCE-FOUND:	REPORT-DIFFERENCE (IPL)

FIGURE 18. Definition of the MATCH-DIFF-METHOD which matches two data structures for all of the differences between them.

Name and Kind of Method	Sub-Methods
REDUCE-METHOD (SEQUENTIAL)	SET-CONTEXT (IPL)
	SELECT-OPERATOR
	APPLY the operator selected to the object of the CURRENT-GOAL (GOAL-SCHEMA)
	REPORT-RESULT (IPL)
SELECT-OPERATOR (SEQUENTIAL, repeat on FAILURE)	FIND-OPERATOR (IPL)
	APPLY-DESIRABILITY-FILTER (IPL)

FIGURE 19. The definition of the REDUCE-METHOD for achieving a REDUCE GOAL.

Name and Kind of Method	Sub-Methods
FORM-OPERATOR-METHOD (SEQUENTIAL)	SET-CONTEXT (IPL)
	EXECUTE-PRETESTS (IPL)
	MATCH-DIFF-METHOD
	SELECT-DIFFERENCE (IPL)
	CONTINUE
CONTINUE (SIGNAL-LIST)	NO-DIFFERENCE: PRODUCE-RESULT (IPL)
	DIFFERENCE-FOUND: MODIFY-AND-APPLY
MODIFY-AND-APPLY (SEQUENTIAL)	REDUCE the difference selected on the object of the APPLY GOAL (GOAL-SCHEMA)
	APPLY the operator of the CURRENT-GOAL to the result of the previous method (GOAL-SCHEMA)
	REPORT-RESULT (IPL)

FIGURE 20. The definition of the FORM-OPERATOR-METHOD for achieving an APPLY GOAL whose operator is a FORM-OPERATOR.

Name and Kind of Method	Sub-Methods
FORM-OPERATOR-TO-SET-METHOD (SEQUENTIAL)	SELECT an object from the SET of the CURRENT-GOAL (GOAL-SCHEMA) APPLY the operator of the CURRENT-GOAL to object selected (GOAL-SCHEMA) REPORT-RESULT (IPL)

FIGURE 21. The definition of the FORM-OPERATOR-TO-SET-METHOD for achieving an APPLY GOAL whose operator is a FORM-OPERATOR and whose input object is a SET of objects.

Name and Kind of Method	Sub-Methods
SET-OPERATOR-METHOD (SEQUENTIAL)	SELECT an operator whose input form is similar to the object of the CURRENT-GOAL (GOAL-SCHEMA) APPLY the operator selected to the object of the CURRENT-GOAL (GOAL-SCHEMA) REPORT-RESULT (IPL)

FIGURE 22. The definition of SET-OPERATOR-METHOD for achieving an APPLY GOAL whose operator is a SET of FORM-OPERATORs.

Name and Kind of Method	Sub-Methods
TWO-INPUT-OPERATOR-METHOD (SEQUENTIAL)	SELECT the input form which is more similar to the object of the CURRENT-GOAL (GOAL-SCHEMA)
	APPLY the input form selected to the object of the CURRENT-GOAL (GOAL-SCHEMA)
	SELECT an object similar to the second input form (GOAL-SCHEMA)
	APPLY the second input form to the object selected (GOAL-SCHEMA)
	REPORT-RESULT (IPL)

FIGURE 23. The definition of the TWO-INPUT-OPERATOR-METHOD for achieving an APPLY GOAL whose operator is a FORM-OPERATOR which has two inputs.

Name and Kind of Method	Sub-Methods	
MOVE-OPERATOR-METHOD (SEQUENTIAL)	SET-CONTEXT (IPL)	
	APPLY-FEASIBLE-OPERATOR	
APPLY-FEASIBLE-OPERATOR (SEQUENTIAL, repeat on FAILURE)	SPECIFY-VARIABLES (IPL)	
	EXECUTE-PRETESTS-AND-TEST-LEGALITY-OF-MOVES (IPL)	
	SELECT-DIFFERENCE (IPL)	
	APPLY-OPERATOR-IF-POSSIBLE	
APPLY-OPERATOR-IF-POSSIBLE (SIGNAL-LIST)	NO-DIFFERENCE:	APPLY-OPERATOR
	DIFFERENCE-FOUND:	MODIFY-IF-NOT-TOO-DIFFICULT
MODIFY-IF-NOT-TOO-DIFFICULT (SEQUENTIAL)	TEST-DIFFICULTY-OF-DIFFERENCE (IPL)	
	REDUCE the difference selected on the object of the CURRENT-GOAL (GOAL-SCHEMA)	
	APPLY the operator of the CURRENT-GOAL to the result of the previous method (GOAL-SCHEMA)	
	REPORT-RESULT (IPL)	
APPLY-OPERATOR (SEQUENTIAL)	APPLY-MOVES-AND-POST-TESTS (IPL)	
	RECORD-RESULT (IPL)	

FIGURE 24. The definition of the MOVE-OPERATOR-METHOD for achieving an APPLY GOAL whose operator is a MOVE-OPERATOR.

Name and Kind of Method	Sub-Methods
TRANSFORM-SET-METHOD (SEQUENTIAL)	SELECT a member of the SET (GOAL-SCHEMA) TRANSFORM the member selected into the second object of the CURRENT-GOAL (GOAL-SCHEMA) REPORT-RESULT (IPL)

FIGURE 25. The definition of the TRANSFORM-SET-METHOD for achieving a TRANSFORM GOAL whose first object is a SET of objects.

Name and Kind of Method	Sub-Methods
EXPANDED-TRANSFORM-METHOD (SEQUENTIAL)	TRANSFORM the SET of objects that are derived from the initial situation into the desired situation (GOAL-SCHEMA) REPORT-RESULT (IPL)

FIGURE 26. The definition of the EXPANDED-TRANSFORM-METHOD for achieving the TOP-GOAL.

Name and Kind of Method	Sub-Methods	
SELECT-BEST-MEMBERS-METHOD (SEQUENTIAL)	SET-CONTEXT (IPL)	
	SELECT-MEMBERS	
SELECT-MEMBERS (SIGNAL-LIST, repeat on signal change)	BEGIN, SEVERAL-MEMBERS-PASSED:	FIND-NEXT-TEST (IPL)
	TEST-FOUND:	APPLY-TEST (IPL)
	ONLY-ONE-MEMBER-PASSED:	USE-MEMBER-FOR-RESULT (IPL)
	NO-MEMBERS-PASSED:	RESULT-IS-PREVIOUSLY-PASSED-MEMBERS (IPL)

FIGURE 27. The definition of SELECT-BEST-MEMBERS-METHOD for achieving a SELECT GOAL whose SET is small.

CHAPTER IV: THE REPRESENTATION OF TASKS

The purpose of this chapter is to give a straight forward description of the representation of tasks. As in the last chapter, no general issues will be discussed. However, in Chapter V we will come back to the representation of tasks and its interaction with the problem solving techniques of GPS.

The internal representation of a task consists of several different kinds of data structures:

a. objects
b. operators
c. GOALs
d. differences
e. TABLE-OF-CONNECTIONS
f. DIFF-ORDERING
g. details for matching objects
h. miscellaneous information

Each of these are described individually below. The discussion points out which of these data structures appear in the specification of a task and which are generated by the problem solving process.

GPS is encoded in the list processing language, IPL-V (Newell [33]), and the internal representation of all task structures consists of IPL symbols, lists, and list structures. For the most part, data structures are encoded in the description lists of IPL[1]. However, the internal representation will not be described in terms of IPL data structures, but rather in a higher level language (tree structures, sets, etc.) in such a

[1] An IPL description list is a sequence of pairs of IPL symbols. The first symbol in each pair is an attribute and the second symbol in a pair is the value of the attribute of the pair. For example, the description list that represents the GOAL of transforming X1 into X2 is GOAL-TYPE TRANSFORM, FIRST-OBJECT X1, SECOND-OBJECT X2. X1 and X2 are not atomic symbols but are list structures.

way that there is a one-to-one correspondence to the IPL data structures.

To give GPS a task is to provide it, via some external representation, with information contained in the data structure listed above. As was the case in most previous reports on GPS, often it is useful to consider the notion of a task environment--that information common to all tasks of a particular kind, e.g., to all integration tasks. The task environment consists of operators, differences, TABLE-OF-CONNECTIONS, DIFF-ORDERING, details for matching objects and miscellaneous information. The GOALs, the objects and the task environment comprise a particular task. However, we will not need this distinction here since one or at most two tasks per task environment were given to GPS.

The last section of this chapter describes the external representation of tasks to aid the reader in understanding the task specifications in Chapter VI. Since the external representation is very similar to the internal representation, only the semantics of the internal representation is described, and the description of external representation deals mainly with syntax. We note again that those words written with all capital letters correspond, directly, to IPL symbols in GPS. These words, which comprise the basic vocabulary of GPS, are given in Appendix A.

To be concrete, the examples used in this chapter will be drawn from either the missionaries and cannibals task or the integration task. Fig. 28 and Fig. 29 show the heuristic search formulations of these tasks. Fig. 30 and Fig. 31 show the specification of these tasks expressed in the external representation of GPS. Only a few integration operators are given in Fig. 29 and Fig. 31. These are typical and are sufficient for the expression to be integrated.

Generic form of objects:	The number of missionaries at each side of the river, the number of cannibals at each side of the river, and the position of the boat.
Initial situation:	Three missionaries, three cannibals, and the boat at the left bank of the river; nothing at the right bank of the river.
Desired situation:	Three missionaries, three cannibals, and the boat at the right bank of the river; nothing at the left bank of the river.
Operators:	Move x missionaries, y cannibals (x and y are variables) and the boat across the river provided that a. $1 \leq x+y \leq 2$ (the boat must not sink and someone must row.) b. at both banks of the river in the resultant object, either the number of missionaries $\geq$ the number of cannibals, or the number of missionaries = 0. (In the resultant object no missionaries can be eaten.)

FIGURE 28. The heuristic search formulation of the missionaries and cannibals task.

Generic form of operators: Any expression.

Initial situation: $\int t\, e^{t}\, dt$

Desired situation: An expression which does not contain an '$\int$'.

Operators:

a. $\int u^{n}\, du = \frac{u^{n+1}}{n+1}$

b. $\int u^{-1}\, du = \log u$

c. $\int \sin u\, du = -\cos u$

d. $\int \cos u\, du = \sin u$

e. $\int u\, du = 1/2\, u^{2}$

f. $\int e^{u}\, du = e^{u}$

g. $\int (f+g)\, du = \int f\, du + \int g\, du$

h. $\sin u\, du = -d \cos u$

i. $\cos u\, du = d \sin u$

j. $u\, du = 1/2\, d\, u^{2}$

k. $u^{-1}\, du = d \log u$

FIGURE 29. The heuristic search formulation of the integration task.

```
RENAME   (
   LEFT = FIRST
   RIGHT = SECOND
     )
DECLARE     (
   BOAT = ATTRIBUTE
   C = ATTRIBUTE
   B-L = FEATURE
   B-R = FEATURE
   C-L = FEATURE
   C-R = FEATURE
   DESIRED-OBJ = OBJECT-SCHEMA
   FROM-SIDE = LOC-PROG
   FROM-SIDE-TESTS = V-TESTS
   INITIAL-OBJ = OBJECT-SCHEMA
   M = ATTRIBUTE
   M-C-OPR = MOVE-OPERATOR
   M-L = FEATURE
   M-R = FEATURE
   SIDE-SET = SET
   TO-SIDE = LOC-PROG
   TO-SIDE-TESTS = V-TESTS
   X = CONSTANT
   X+Y = EXPRES
   Y = CONSTANT
   0,1,2-SET = SET
   1,2 = SET
```

Figure 30: The specification for GPS of the missionaries and cannibals task.

```
    )
TASK-STRUCTURES     (
   TOP-GOAL = ( TRANSFORM THE INITIAL-OBJ INTO THE DESIRED-OBJ . )
   INITIAL-OBJ = ( LEFT ( M 3 C 3 BOAT YES )
                   RIGHT ( M 0 C 0 ) )
   DESIRED-OBJ = ( LEFT ( M 0 C 0 )
                    RIGHT ( M 3 C 3 BOAT YES ) )
   X+Y = ( X + Y )
   1,2 = ( 1 2 )
   0,1,2-SET = ( 0 1 2 )
   SIDE-SET = ( LEFT RIGHT )
   FROM-SIDE-TESTS = (   1. THE M OF THE FROM-SIDE IS NOT-LESS-THAN
                            THE C OF THE FROM-SIDE ,
                         2. THE M OF THE FROM-SIDE EQUALS 0 . )
   TO-SIDE-TESTS = (  1. THE M OF THE TO-SIDE IS NOT-LESS-THAN
                         THE C OF THE TO-SIDE .
                      2. THE M OF THE TO-SIDE EQUALS 0 . )
   M-C-OPR = ( CREATION-OPERATOR
               $ MOVE X MISSIONARIES AND Y CANNIBALS FROM THE FROM-SIDE TO
                 THE TO-SIDE $
               VAR-DOMAIN
               1. Y IS A CONSTRAINED-MEMBER OF THE 0,1,2-SET ,
                  THE CONSTRAINT IS X+Y IS IN-THE-SET 1,2 ,
               2. X IS A CONSTRAINED-MEMBER OF THE 0,1,2-SET ,
                  THE CONSTRAINT IS X+Y IS IN-THE-SET 1,2 ,
               3. THE FROM-SIDE IS AN EXCLUSIVE-MEMBER OF THE SIDE-SET ,
               4. THE TO-SIDE IS AN EXCLUSIVE-MEMBER OF THE SIDE-SET .
```

Figure 30: (continued)

```
                MOVES
                  1. MOVE THE BOAT OF THE FROM-SIDE TO THE BOAT OF THE TO-SIDE .
                  2. DECREASE BY THE AMOUNT X THE M AT THE FROM-SIDE AND ADD
                     IT TO THE M AT THE TO-SIDE .
                  3. DECREASE BY THE AMOUNT Y THE C AT THE FROM-SIDE AND ADD
                     IT TO THE C AT THE TO-SIDE .
                POST-TESTS
                  1. ARE ANY OF THE FROM-SIDE-TESTS TRUE .
                  2. ARE ANY OF THE TO-SIDE-TESTS TRUE . )
B-L = ( BOAT ON THE LEFT . )
B-R = ( BOAT ON THE RIGHT . )
C-L = ( C ON THE LEFT . )
C-R = ( C ON THE RIGHT . )
M-L = ( M ON THE LEFT . )
M-R = ( M ON THE RIGHT . )
DIFF-ORDERING = ( ( M-R M-L C-R C-L )
                  ( B-R B-L ) )
TABLE-OF-CONNECTIONS = ( ( COMMON-DIFFERENCE M-C-OPR ) )
COMPARE-OBJECTS = ( BASIC-MATCH )
BASIC-MATCH = ( COMP-FEAT-LIST ( M-L C-L B-L ) )
OBJ-ATTRIB = ( M C BOAT )
LIST-OF-VAR = ( FROM-SIDE TO-SIDE X Y )
  )     END
```

Figure 30: (continued)

```
RENAME   (
   LEFT = FIRST
   RIGHT = SECOND
     )
DECLARE   (
   COS = UNARY-CONNECTIVE
   D = UNARY-CONNECTIVE
   DESIRED-OBJ = DESCRIBED-OBJ
   EXP = N-ARY-CONNECTIVE
   INTEGRAL = UNARY-CONNECTIVE
   LOG = UNARY-CONNECTIVE
   SIN = UNARY-CONNECTIVE
   SYMBOL = ATTRIBUTE
   SYN-DIFF = FEATURE
   - = UNARY-CONNECTIVE
   * = N-ARY-CONNECTIVE
   + = N-ARY-CONNECTIVE
     )
LIST     (
   EXPRESSION-1 = ( THE INTEGRAL OF ( T * ( E EXP ( T EXP TWO ) ) * D T ) )
   INTEGRATE = ( 1. ( THE INTEGRAL OF ( ( U EXP N ) * D U ) YIELDS
                    ( ( U EXP ( N + ONE ) ) * ( ( N + ONE ) EXP - ONE ) ) ) ,
                2. ( THE INTEGRAL OF ( ( U EXP - ONE ) * D U ) YIELDS
                   LOG U ) ,
                3. ( THE INTEGRAL OF ( SIN U * D U ) YIELDS - COS U ) ,
                4. ( THE INTEGRAL OF ( COS U * D U ) YIELDS SIN U ) ,
                5. ( THE INTEGRAL OF ( U * D U ) YIELDS ( ( U EXP TWO ) *
```

Figure 31: The specification for GPS of the task of integrating $\int te^{t^2} dt$.

```
                    ( TWO EXP - ONE ) ) ) ,
                 5. ( THE INTEGRAL OF ( ( E EXP U ) * D U ) YIELDS
                    ( E EXP U ) ) ,
                 7. ( THE INTEGRAL OF ( ( F + G ) * D U ) YIELDS ( THE INTEGRAL
                    OF ( F * D U ) + THE INTEGRAL OF ( G * D U ) ) ) )
   DIFFERENTIATE = ( 1. ( ( SIN U * D U ) YIELDS - U COS U ) ,
                     2. ( ( COS U * D U ) YIELDS U SIN U ) ,
                     3. ( ( U * D U ) YIELDS ( U TWO EXP - ONE ) *
                        U ( U EXP TWO ) ) ) ,
                     4. ( ( ( U EXP - ONE ) * D U ) YIELDS U LOG U ) )
   EXPRESSION-2 = ( THE INTEGRAL OF ( ( ( ( SIN ( C * T ) EXP TWO ) *
                  COS ( C * T ) ) * ( T EXP - ONE ) ) * D T ) )
     )
TASK-STRUCTURES   (
   TOP-GOAL = ( TRANSFORM EXPRESSION-1 INTO THE DESIRED-OBJ . )
   DESIRED-OBJ = ( SUBEXPRESSION-TESTS
                   THE SYMBOL DOES NOT-EQUAL INTEGRAL . )
   SYM-DIFF = ( SYMBOL )
   TABLE-OF-CONNECTIONS = ( 1. ( SET-SIZE DIFFERENTIATE )
                            2. ( SYM-DIFF INTEGRATE ) )
   DIFF-ORDERING = ( 1. SYM-DIFF
                     2. SET-SIZE )
   LIST-OF-OPR = ( INTEGRATE DIFFERENTIATE )
   LIST-OF-VAR = ( F G U N )
   )
   END
```

Figure 31: (continued)

A. OBJECTS

In GPS there are two basic representations for objects which are described below.

OBJECT-SCHEMAS

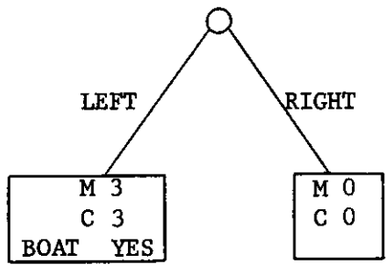
An OBJECT-SCHEMA is a tree structure, encoded in IPL description lists, which represents an object. Each node of the tree structure can have an arbitrary number of branches leading from it to other nodes. In addition to branches, each node can have a local description given by an arbitrary number of attribute-value pairs. The tree structure in Fig. 32.a, for example, represents the initial situation in the missionaries and cannibals task. In Fig. 32.a the node to which the LEFT[2] branch leads represents the left bank of the river and the node to which the RIGHT branch leads represents the right bank of the river. The local description at the node which the LEFT branch leads to indicates that there are three missionaries, three cannibals and a boat at that bank of the river.

GPS knows[3] the generic form of OBJECT-SCHEMAS. In particular, it knows the names of the branches FIRST, SECOND, etc. and thus knows that all OBJECT-SCHEMAs have the form illustrated in Fig. 32.b. GPS also knows the form of the local descriptions of the nodes and can generate the

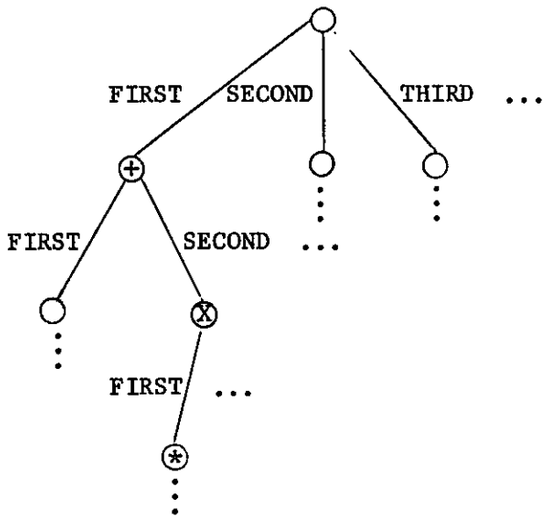
2LEFT is not part of the basic vocabulary of GPS, but it is capitalized because it is defined in the task specification (Fig. 30).

3We say that GPS knows something when the programmer attached special significance to it in constructing the code that defines GPS. Hence, GPS knows the generic form of OBJECT-SCHEMAs because it is a programming convention of those routines that process OJBECT-SCHEMAs. However, GPS does not understand the details of a particular task; they are defined in the task specification in terms of the things that GPS knows, listed in Appendix A.

(a)



(b)



(c)

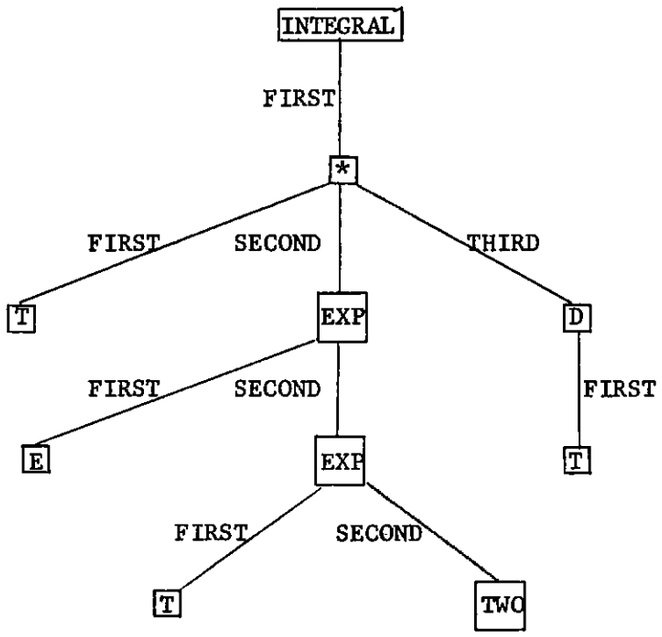


FIGURE 32. (a) and (c) are the tree structure representations of the initial situations of the missionaries and cannibals task and an integration task, respectively. (b) is the generic form of an OBJECT-SCHEMA.

attribute-value pairs of a node because ATTRIBUTEs are a special class of symbols. For example, GPS knows that the M, C, and BOAT in Fig. 32.a are ATTRIBUTEs because they are so declared in the task specification (Fig. 30).

LOC-PROGs (LOCation-PROGrams) are a special class of data structures used to refer to the nodes of an OBJECT-SCHEMA. They consist of an ordered list of the branches on the path between two nodes. The name of a LOC-PROG, which may be used in the specification of tasks, is the word formed by connecting all of the branches listed on it with hyphens. LOC-PROGs designate one node of an OBJECT-SCHEMA relative to another node; i.e., they are functions of one argument--a node of an OBJECT-SCHEMA. For example, the LOC-PROG, SECOND-FIRST of the node containing a '+' in Fig. 32.b designates the node containing a '*'. The name of a LOC-PROG that designates an immediate subnode of a node is the name of the branch which leads to the node. For example, the LOC-PROG, SECOND, of the node containing a '+' in Fig. 32.b designates the node containing a 'X'. TOP-NODE is the LOC-PROG that designates the top most node of an OBJECT-SCHEMA. In Fig. 32.a, LEFT is the new name assigned to FIRST in the task specification (Fig. 30).

The given object in the integration task in Fig. 29 is represented by the tree structure in Fig. 32.c. In this example, it is assumed that each node has two ATTRIBUTEs--SIGN and SYMBOL--but only their values are shown in Fig. 32.c. The usual convention that a missing sign signifies a plus is adopted in this example. Since all of the signs are positive none appear in Fig. 32.c.

As mentioned in Chapter II, often there is more than one given

object or one desired object in a problem. The use of variables in OBJECT-SCHEMAs allows a class of objects to be represented as a single structure. For example, the OBJECT-SCHEMA,

$$\int e^{u}\, du$$

where u is a free variable, represents a large class of objects. All members of the class have the same form but different values for u. Two objects that are members of the class are

$$\int e^{t^2}\, dt^2$$

and

$$\int e^{\sin t}\, d \sin t.$$

GPS assumes that all OBJECT-SCHEMAs may contain variables and is prepared to process them as a class of objects. GPS can recognize the variables in OBJECT-SCHEMAs because the task specification indicates which symbols are variables.

OBJECT-SCHEMAs can only represent those classes of objects whose members all have the same form. One way to represent classes of objects whose forms are different is a list of OBJECT-SCHEMAs. In GPS the initial situation, but not the desired situation, can be a list of OBJECT-SCHEMAs[4]. But either the initial situation or the desired situation, both of which are given in the task specification, can be an OBJECT-SCHEMA. All of the objects generated during problem solving are OBJECT-SCHEMAs.

4This restriction, and several others, arises because the methods of GPS cannot deal with a more general case. Sometimes (although not always) the restriction could be lifted by the addition of new methods. However, since we wish to keep the problem solving character of GPS constant, we will not consider the addition of new methods.

DESCRIBED-OBJs

A DESCRIBED-OBJ is a list of constraints that represents the class of objects, each of which satisfies all of the constraints. The desired object in the integration task (Fig. 29) can be represented by a DESCRIBED-OBJ as the single constraint which must be satisfied at each node:

The SYMBOL does not equal $\int$.

Each constraint in a DESCRIBED-OBJ is a TEST which is a data structure consisting of a RELATION, and several arguments (in most cases, two). A RELATION is a Boolean function of several arguments and is a special type of symbol. In the previous example, the TEST is:

RELATION is NOT-EQUAL;
first argument is SYMBOL,
second argument is $\int$.

GPS recognizes NOT-EQUAL as a RELATION and understands the semantics of all RELATIONs. (GPS currently knows fifteen RELATIONs, whose definitions are given in Appendix A, and only these can be used, unless new ones are added.) On the other hand, $\int$ is peculiar to the integration task and GPS treats it as a symbolic CONSTANT. SYMBOL is also peculiar to integration, but it is declared in the task specification to be an ATTRIBUTE. Thus, GPS treats SYMBOL in the integration task in the same way as M (missionaries) in the missionaries and cannibals task.

An argument of a TEST is either a constant (i.e., not a function of objects, such as a CONSTANT or SET) or a FEATURE of an object. A FEATURE, which is a means of referring to a feature of an arbitrary object, is a function of one argument--a node of an object. A FEATURE is specified

by an ATTRIBUTE and a LOC-PROG; the value of the FEATURE is the value of the ATTRIBUTE of the node designated by the LOC-PROG. (FEATUREs always have values, but a possible value is UNDEFINED.) For example, the FEATURE,

M LEFT,

of the OBJECT-SCHEMA in Fig. 32.a has the value, 3. LOC-PROGs and ATTRIBUTEs, like FEATUREs, are functions of one argument--a node of an object. Fig. 33 shows how FEATUREs are evaluated.

Since a FEATURE is a function of an implied node of an object, a TEST is also a function of an implied node of an object because it may contain a FEATURE. In a DESCRIBED-OBJ, there are in general two kinds of TESTs

a. those whose implied node is the TOP-NODE of the object;
b. those whose implied node is every node of the object.

Each of the latter set of TESTs is evaluated for each node of the object as the implied node, and the TEST is true only if it is true for each evaluation.

Only the desired situation can be represented as a DESCRIBED-OBJ; the initial situation and the objects generated by the problem solving process cannot be represented as DESCRIBED-OBJ.

B. OPERATORS

In GPS there are two different types of operators, which are discussed below.

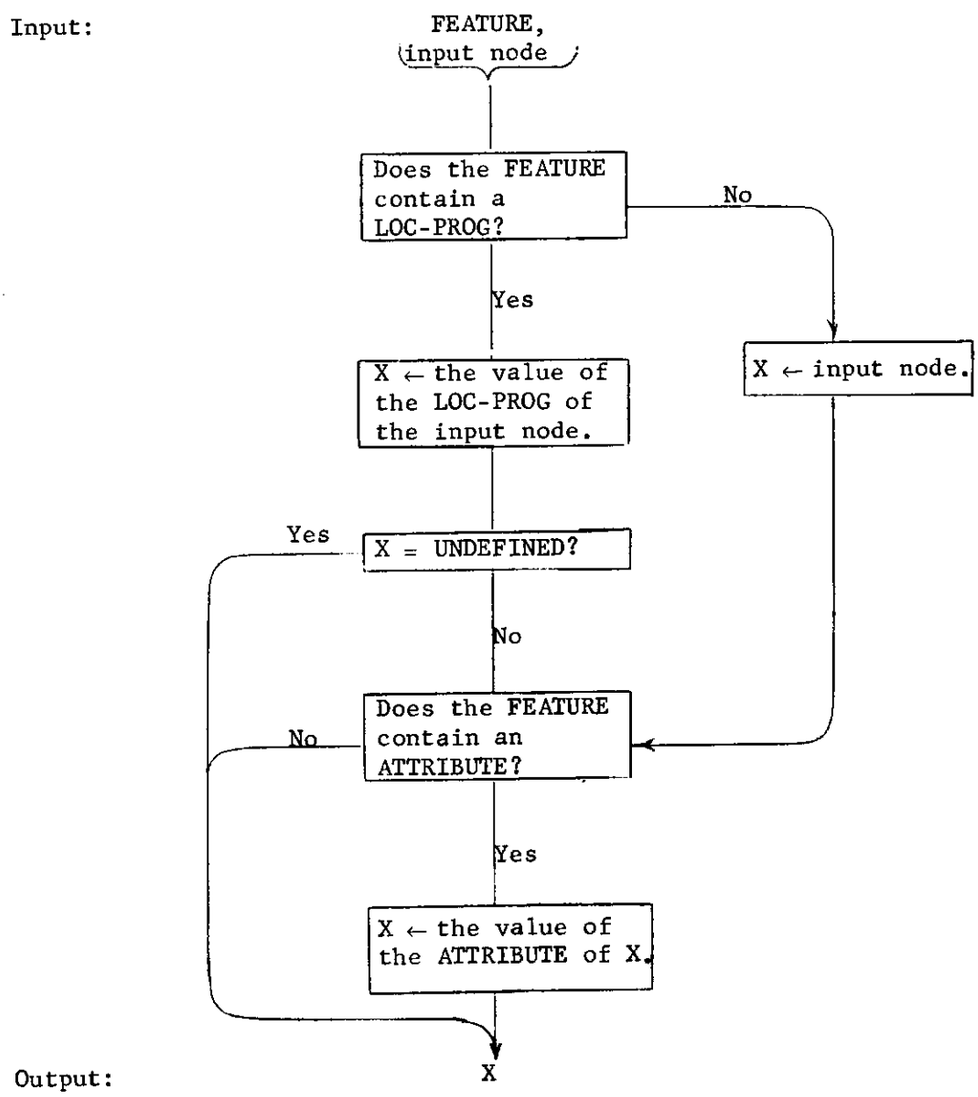


FIGURE 33. The flow-chart for the evaluation of a FEATURE of a node of an OBJECT-SCHEMA.

FORM-OPERATORS

A simple way to express an operator is to state the form of its input object and the form of its resultant object. For example, the first operator in Fig. 29 can be expressed as, assuming u as a variable,

input object form: $\int u^n du$
resultant object form: $\frac{u^{n+1}}{n+1}$.

Such an expression will be called a FORM-OPERATOR. Both forms are OBJECT-SCHEMAs and thus represent classes of objects. The above FORM-OPERATOR can be applied to any object in the class of objects represented by the input OBJECT-SCHEMA. The resultant object is obtained by substituting for u in the resultant OBJECT-SCHEMA, the value used for u in the input OBJECT-SCHEMA.

All of the operators in the integration task in Fig. 29 can be expressed as FORM-OPERATORs by expressing the left side of the equation and the right side of the equation as the input OBJECT-SCHEMA and the resultant OBJECT-SCHEMA, respectively. However, FORM-OPERATORs are not an adequate means for expressing the operators of some tasks, e.g., the missionaries and cannibals task.

MOVE-OPERATORs

A MOVE-OPERATOR represents an operator by a group of TESTs and a group of TRANSFORMATIONs. The TRANSFORMATIONs describe how the resultant object differs from the input object, and the TESTs must be satisfied for the operator to be feasible. In the missionaries and cannibals task, for example, the operator in Fig. 28 can be stated as a MOVE-OPERATOR. The TESTs of the operator insure that nothing chaotic happens, such as missionaries being eaten, and the TRANSFORMATIONs describe how the object changes

when people cross the river in the boat. Before giving a complete statement of the MOVE-OPERATOR representation of the operator in Fig. 29, TRANSFORMATIONs are defined.

A TRANSFORMATION is a data structure consisting of an OPERATION and several arguments. An OPERATION is a special type of symbol and is a function of several arguments. A typical TRANSFORMATION which might appear in the missionaries and cannibals task is:

OPERATION is MOVE;
first argument is BOAT LEFT;
second argument is BOAT RIGHT.

The meaning of this TRANSFORMATION is:

a. Find the value of the BOAT at the LEFT side of the river.
b. Make the value of the BOAT at the RIGHT side of the river equal to the result of step a.
c. Remove the BOAT at the LEFT side of the river.

If the result of a is UNDEFINED, the TRANSFORMATION is not applicable to the object, and the operator is infeasible.

GPS knows the meaning of MOVE and all other OPERATIONs in the same way that it understands the RELATIONs. (There are six different OPERATIONs which are defined in Appendix A.) GPS also recognizes that the two arguments of the above TRANSFORMATION are FEATUREs. Although both arguments of MOVE must be FEATUREs, in general, only the second argument of an OPERATION must be a FEATURE.

The TRANSFORMATIONs in Fig. 34.a describe how an object is modified when X missionaries (M) and Y cannibals (C) go from the FROM-SIDE of the river to the TO-SIDE of the river. Since X, Y, FROM-SIDE,

(a) 1. MOVE (BOAT FROM-SIDE , BOAT TO-SIDE)

2. DECREASE (M FROM-SIDE , M TO-SIDE , X)

3. DECREASE (C FROM-SIDE , C TO-SIDE , Y)

(b) 1. One is TRUE :

NOT-GREATER-THAN (C LEFT , M LEFT)

EQUALS (M LEFT , 0)

2. One is TRUE :

NOT-GREATER-THAN (C RIGHT , M RIGHT)

EQUALS (M RIGHT , 0)

(c) 1. EXCLUSIVE-MEMBER (FROM-SIDE , {LEFT , RIGHT})

2. EXCLUSIVE-MEMBER (TO-SIDE , {LEFT , RIGHT})

3. CONSTRAINED-MEMBER (X , {0 , 1 , 2} ,
IN-THE-SET (X + Y , {1 , 2}))

4. CONSTRAINED-MEMBER (Y , {0 , 1 , 2} ,
IN-THE-SET (X + Y , {1 , 2}))

FIGURE 34. The operator which moves X missionaries (M), Y cannibals (C) and the BOAT from the FROM-SIDE to the TO-SIDE. (a) is the TRANSFORMATIONs of the operator; (b) is the TESTs which the resultant object of the operator must satisfy; (c) is the TESTs which constrain the legitimate values of the variables.

and TO-SIDE are all variables, these three TRANSFORMATIONs represent all possible ways that missionaries and cannibals can cross the river. The second TRANSFORMATION in Fig. 34.a means[5]:

a. Decrement the number of missionaries at the FROM-SIDE by X.

b. Increment the number of missionaries at the TO-SIDE by Y.

If the number of missionaries at the FROM-SIDE is less than X, the TRANSFORMATION is not applicable and the operator is infeasible.

TRANSFORMATIONs are functions of nodes of objects because they contain FEATUREs which are functions of nodes of objects. But unlike a TEST, each TRANSFORMATION is a function of two object nodes--a node of the input object and the corresponding node of the resultant object. The input object node is used to designate parameters of the TRANSFORMATIONs, and the TRANSFORMATIONs are performed relative to the resultant object node. In the first TRANSFORMATION in Fig. 34.a, for example, the value of the BOAT at the FROM-SIDE of the input object is found. Then this value is placed in the resultant object as the value of the BOAT at the TO-SIDE. Finally, the ATTRIBUTE, BOAT, and its value are removed from the FROM-SIDE of the resultant object. This action does not involve the input object. In this particular example, the value has little significance because the ATTRIBUTE, BOAT, always has the same value--YES. But, in general, an ATTRIBUTE may have many different values and the value which is "moved" is its value in the input object.

[5]It would seem that move would be a better name than DECREASE. But MOVE is a different OPERATION and DECREASE results in a readable statement in the task specification. See the MOVES in the M-C-OPR in Fig. 30.

The constraints in Fig. 34.b represent the class of objects in which no missionaries can be eaten. The form of these constraints is the conjunction of two disjunctive sets of TESTs. Normally in a set of TESTs, which represents a class of objects, the logical connective of the tests is a conjunction. That is, all objects which are members of the class, satisfy all of the TESTs. But the set of TESTs can be divided into disjunctive subsets by using TRUE[6] which GPS recognizes and understands. TRUE is a RELATION on one argument--a set of TESTs. It is the only RELATION that has a set of TESTs as an argument and is processed somewhat differently than other RELATIONs.

All legal missionaries and cannibals objects must satisfy the constraints in Fig. 34.b; else missionaries would be eaten. Consequently, both the input object and the resultant object of the operator must satisfy the constraints in Fig. 34.b. But, if all objects produced by the application of the operator satisfy the constraints in Fig. 34.b, all of the input objects to the operator will necessarily satisfy the constraints because the input object is either the initial situation or a result of a previous operator application. Thus, the constraints in Fig. 34.b need only be satisfied by the resultant object. This fact is indicated by a syntactic device described in the last section of this chapter.

The variables in MOVE-OPERATORs are not universally quantified but can only take on certain values. The TESTs in Fig. 34.c must be satisfied if the variables in the TRANSFORMATIONs in Fig. 34.a are to have legitimate values. The first two TESTs in Fig. 34.c insure that both TO-SIDE and

---

6Although TRUE is a strange name for this RELATION (or would be more suggestive), it results in a readable statement. See POST-TESTS of the M-C-OPR in Fig. 30.

FROM-SIDE are either LEFT or RIGHT but that they are different. A TEST whose RELATION is EXCLUSIVE-MEMBER is true if the first argument is in the set designated by the second argument, and if no other "exclusive member" of the set has the same value as the first argument. Thus, the concept of an exclusive member is an alternate way to state that both α and β are in a set and α is unequal to β.

The third and fourth TESTs in Fig. 34.c insure that the number of people in the boat is legitimate. A TEST whose RELATION is CONSTRAINED-MEMBER is true, if the third argument, which must be a TEST, is true and if the first argument is in the set designated by the second argument. CONSTRAINED-MEMBER and its negation are the only RELATIONs which have a TEST as an argument. Fig. 34 is the MOVE-OPERATOR representation of the operator in Fig. 28 without the syntax required by the external representation.

In general, a MOVE-OPERATOR is a set of TRANSFORMATIONs, and three sets of TESTs. The operator is feasible if and only if all of the TESTs (with the exception of some in a disjunctive subset) are satisfied and the TRANSFORMATIONs are applicable. Two of the sets of TESTs are stated relative to the node of the input object to which the operator is applied. One of these (indicated syntactically by VAR-DOMAIN) must be satisfied in order for the variables in the operator to have legitimate values. The other (indicated syntactically by PRETESTS) constrains the class of input objects for which the operator is feasible. The third set of TESTs (POST-TESTS) must be satisfied by the resultant object. Any of these sets may be empty, in which case it is, by convention, satisfied.

C. GOALS

A GOAL is a data structure consisting of the information necessary for problem solving context. For example, a GOAL has, as the values of attributes, the names of its supergoals, the names of subgoals, its type, its objects, its operator, its difference, etc. The only GOAL that appears in the specification of a task is the TOP-GOAL--the statement of the problem. The TOP-GOAL of all of the tasks in Chapter VI has the same form: TRANSFORM one object into another. The type and the objects of the TOP-GOAL are given in the task specification; all of the other information is generated internally by GPS. Although the TOP-GOAL could be a different type of GOAL, e.g., REDUCE, this situation has not arisen.

D. DIFFERENCES

A difference, which GPS detects in matching two objects, is a data structure consisting of a difference type, the value of the difference, and the name of the node where the difference was detected. For example, in matching the given situation to the desired situation in the missionaries and cannibals task, GPS would find the following differences:

Difference type is M, LEFT;
difference value is -3;
difference location is TOP.

Difference types are task dependent and must be defined for each task. But differences are not given in the task specification because they are generated during problem solving.

Currently, all difference types are FEATUREs. Although a richer representation for difference types would be desirable, FEATUREs are

adequate for the tasks discussed in Chapter VI[7].

E. TABLE-OF-CONNECTIONS

TABLE-OF-CONNECTIONS is a data structure that associates with each type of difference a list of operators. Each operator on a list has the capability of reducing the difference type with which the list is associated. Thus, TABLE-OF-CONNECTIONS is a way to give GPS, exogenously, information about the properties of operators. The operators in the TABLE-OF-CONNECTIONS can be either FORM-OPERATORs or MOVE-OPERATORS.

F. DIFF-ORDERING

DIFF-ORDERING is a list of difference types and/or groups of difference types which are ordered according to difficulty. GPS considers the difference types at the top of the list to be the most difficult to alleviate and those at the bottom the easiest. All difference types within a group on the list are considered equally difficult. DIFF-ORDERING is a means to give GPS information about the nature of the difference types.

G. COMPARE-OBJECTS

The MATCH-DIFF-METHOD (Fig. 18) subdivides a data structure into parts and detects differences (if any exist) between corresponding parts. COMPARE-OBJECTS is a data structure that specifies for the MATCH-DIFF-METHOD how two OBJECT-SCHEMAs should be matched. There are two options for subdividing OBJECT-SCHEMAs into parts: match each pair of correspond-

7The logic task that was given to previous versions of GPS (see page 30) required more complex differences such as decrease the number of occurrences a term.

ing nodes or match the two OBJECT-SCHEMAs in toto. COMPARE-OBJECTS specifies one of these options as well as the types of differences to be detected.

If the desired situation is a DESCRIBED-OBJ, COMPARE-OBJECTS is not a required part of the task representation because the information is contained implicitly in the process which compares a DESCRIBED-OBJ to an OBJECT-SCHEMA.

H. MISCELLANEOUS INFORMATION

Besides the foregoing, other information must be represented. OBJ-ATTRIB is a list of the ATTRIBUTEs of the task. LIST-OF-VAR is a list of the symbols which should be interpreted as variables. LIST-OF-OPR is a list of the FORM-OPERATORS of the task.

The task description also contains the type of both symbols and data structures which are peculiar to a particular task, For example, in the missionaries and cannibals task the initial situation must be declared an OBJECT-SCHEMA and BOAT must be declared an ATTRIBUTE. All symbols and data structures have types associated with them but the type of some symbols, e.g., LOC-PROGs and RELATIONs are part of GPS's basic knowledge.

Immediate operators, which are used in the MATCH-DIFF-METHOD (Fig. 18), and selection criteria, which are associated with SELECT GOALs, are represented as IPL programs. Although they are task dependent information, they do not appear in the task specifications because of insufficient facilities in the external representation.

I. EXTERNAL REPRESENTATION OF GPS

Since the emphasis of the research reported here is on the internal representation of tasks, it was designed without any consideration of how

it might be expressed in readable language. The external representation and the translator for it were then designed so that a task expressed in the external representation would be readable, and so that minor modifications and extensions of the external representation could easily be made. Consequently, the external language for specifying tasks and the translator have no theoretical significance and the language is only partially readable.

General Structure of the External Representation

A task description in the external representation is a string of words, and the final word in the string is END. Space is used to delimit words. A word, then, is any string of characters that does not contain a space but is preceded and followed by a space.

Comments, which are ignored, can be inserted in the string of words at any occurrence of a space. $ is used to denote comments. A comment is a $ followed by any string of characters not containing a $ followed by a $. All comments must be preceded and followed by a space.

A task description contains meta-words and text-words. The meta-words are instructions to the translator on how to interpret the text-words. All of the meta-words are listed in Appendix A, section 1, along with their function.

The most commonly used meta-words are (and). They serve their usual function of delimiting a group of words. Throughout the translating process) is matched to the corresponding (so that they can be nested. Although the use of parentheses is a very powerful punctuation device, extensive use of them makes the text unreadable. An attempt was made to give the text a syntactic structure which limited the occurrences of parentheses. In addition tc parentheses, periods, and commas are used to

delimit groups of words.

All of the remaining meta-words determine the mode of translation. Text-words are translated under the current mode of translation and the current mode can only be changed by the occurrence of certain meta-words, each of which is discussed below.

RENAME

The text-words in the basic vocabulary of GPS, which are given in Appendix A, appear in task specifications and are replaced by their corresponding IPL symbols in translation. For convenience, these words can be assigned new names in the RENAME mode of translation. The format of such assignment statements is

W1 = W2.

The word W1 becomes associated with the IPL symbol which was previously associated with W2. W2 is free to be used for a different purpose. For example, in the missionaries and cannibals task specification, FIRST is assigned the new name LEFT.

SKIP WORDS

The basic strategy of the translator is to process key words and ignore all others. All of the text-words translated in the SKIP-WORDS mode are designated as words which should be ignored. The words that are normally ignored are listed in Appendix A, section 3.

DECLARE

All text-words that are not assigned an internal symbol, are not ignored, and are not in the basic vocabulary (e.g., ATTRIBUTE, OBJECT-SCHEMA, etc.) are assigned a type. In the DECLARE mode of translation a word is assigned a type by a statement of the form,

W1 = W2.

This means that the word W2 is assigned to be the type of the word W1. For example, in the missionaries and cannibals task, BOAT is assigned the type ATTRIBUTE. If a word is not explicitly assigned a type in the DECLARE mode, it is, by convention, assigned the type CONSTANT.

TASK-STRUCTURES

All of the data structures of a task, e.g., FEATUREs, OBJECT-SCHEMAs, etc., are translated in the TASK-STRUCTURES mode after the words have been assigned types. The format for data structures is

W1 = (W2 W3...WN).

The data structure's name is W1 and W2...WN are the words which comprise the structure. The parentheses must be "matching parentheses"; i.e., other pairs of parentheses can occur in W2...Wn.

The name of all structures must be assigned a type prior to translating it as a data structure. Associated with certain types are the forms of their data structures and the structures are translated according to these forms. For example, the missionaries and cannibals task (Fig. 30) contains the data structure

M-L = (THE M AT THE LEFT.)

Prior to the occurrence of this structure, M-L was assigned the type FEATURE. The form of a FEATURE has two "slots":
One is for an ATTRIBUTE and the other is for a LOC-PROG, and neither "slot" need be filled. In constructing M-L, the translator copies the form of a FEATURE and fills the "ATTRIBUTE slot" with M, since it was declared to be an ATTRIBUTE, and the "LOC-PROG slot" with LEFT, since it is the new name of FIRST which is a LOC-PROG. (AT and THE are ignored.) It then assigns this data structure to be M-L.

Most data structures are quite similar to OBJECT-SCHEMAs in that they are tree structures. Fig. 35 shows a MOVE-OPERATOR in the internal tree structure form. The tree representation is used for MOVE-OPERATORs in order to keep the internal representation of data homogeneous. However, it seems more natural to consider MOVE-OPERATORS as groups of TESTs and TRANSFORMATIONs, rather than as a tree.

The main work of the translator is to convert the linear text into the hierarchial structures of the internal representation. This conversion is done by filling slots in the forms, because the forms have the hierarchial structure built into them. There is a provision for adding new forms without modifying the translator.

The formal syntax of the language will not be given because it is rather complicated. An informal description of the syntax will be given by describing the standard forms of the data structures. However, most of these standard forms must be modified slightly in certain cases, for example, TEST usually has two arguments, but when the RELATION is CONSTRAINED-MEMBER it has a third argument. Thus, there are some syntactic devices for modifying forms, such as a means for inserting symbols in standard forms and punctuation, e.g., . for terminating structures. Any exceptions to the standard forms will be noted in the Appendix A, section 2, under the words which are responsible for the modification of the forms, e.g., the definition of CONSTRAINED-MEMBER designates the fact that it is a RELATION on three argument.

The notation used for a form is a list of items. Each member of the list will be assigned a label such as a, b, etc. If the order of the list is important, it will be designated as an ordered form; otherwise, it is in unordered form. Certain items in the forms may be

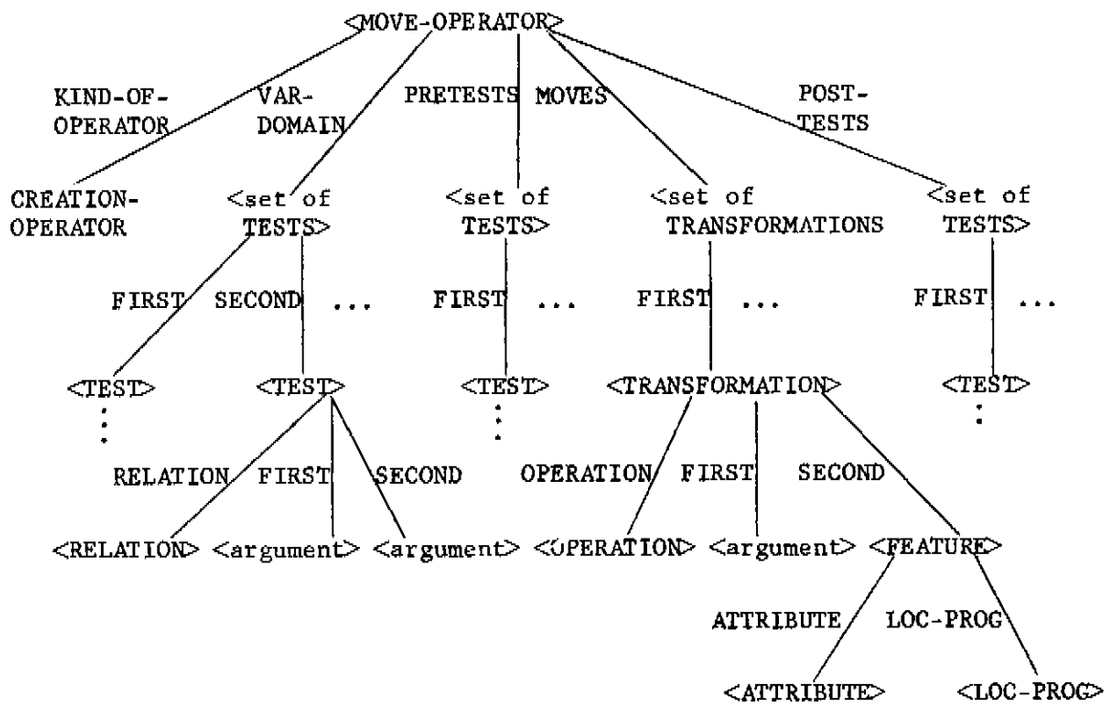


FIGURE 35. The tree structure representation of a MOVE-OPERATOR.

optional; i.e., they may occur but are not required. Unless otherwise stated, items are not optional. The meta symbols, '<' and '>' are used to denote classes, e.g., <ATTRIBUTE> means any word which has been assigned the type ATTRIBUTE.

TOP-GOAL. TOP-GOAL has the ordered form

a. TRANSFORM
b. <OBJECT-SCHEMA or list of OBJECT-SCHEMAS>
c. <OBJECT-SCHEMA or DESCRIBED-OBJ>.

Both b and c are names and not data structures. For example, b can be the name of a list of OBJECT-SCHEMAs but not the names of several OBJECT-SCHEMAs.

FEATURES. A FEATURE has the unordered form

a. <ATTRIBUTE>
b. <LOC-PROG>

Both a and b are optional. M-L in Fig. 30 is an example of a FEATURE. LEFT is b and M is a. The other words are ignored.

SETS. A SET is an ordered list of words. Although SETs are ordered the order is usually unimportant. For example, IN-THE-SET does not depend on the order of the elements of a SET. SIDES in Fig. 30 is the set of two elements, LEFT and RIGHT.

EXPRES. An EXPRES (EXPRESsion) is a list of numbers or variables that stand for numbers separated by +. This elementary form of an arithmetic expression could easily be generalized, but it is sufficient for the tasks encountered.

TESTs. A TEST has the ordered form

a. <argument>
b. <RELATION>
c. <argument>

Each argument is either a FEATURE, a CONSTANT, a SET, an EXPRES, or a variable that stands for a CONSTANT. In the first TEST following VAR-DOMAIN in Fig. 30, a is FROM-SIDE, b is EXCLUSIVE-MEMBER, and c is SIDES. The other words are ignored.

TRANSFORMATIONs. The ordered form of a TRANSFORMATION is

a. <OPERATION>
b. <argument>
c. <FEATURE>

In general, the argument can be of the same type as the argument of a TEST, but certain OPERATIONs require that the argument must be a FEATURE. In the first TRANSFORMATION following MOVES in Fig. 30, a is MOVE, b is the FEATURE,

FROM-SIDE BOAT

and c is the FEATURE,

TO-SIDE BOAT.

MOVE-OPERATORs. MOVE-OPERATOR has the unordered form

a. <kind of operator>
b. VAR-DOMAIN <list of TESTs>
c. PRETESTS <list of TESTs>
d. MOVES <list of TRANSFORMATIONs>
e. POSTESTS <list of TESTs>

In all of the examples in Chapter VI, a is a CREATION-OPERATOR, because the resultant object is always to be a new object. An alternative value for a is MODIFICATION-OPERATOR, which would indicate that the resultant

object is the input object, modified. MODIFICATION-OPERATORS never occur because GPS assumes that after an object is generated it does not get destroyed. Only a and d must exist; b, c, and e are optional.

The first argument of each TEST associated with VAR-DOMAIN is assumed to be a variable which stands for itself. Often such a variable could be a FEATURE as well. For example, in Fig. 30 the first argument of the first TEST of VAR-DOMAIN is FROM-SIDE. The intent is that the value of this argument is either LEFT or RIGHT and not a node which represents the bank of the river in an object.

The TESTs following POSTESTS is stated relative to the resultant object; the others are stated relative to the input object. M-C-OPR in Fig. 30 is an example of a MOVE-OPERATOR in which c is missing.

DESCRIBED-OBJs. A DESCRIBED-OBJ has the unordered form,

a. TEX-DESCRIPTION <set of TESTs>
b. SUBEXPRESSION-TESTS <set of TESTs>

Both a and b are optional. DESIRED-OBJ in Fig. 31 is an example of a DESCRIBED-OBJ. Since there are no TESTs peculiar to the TOP-NODE in it, a is missing. The TESTs in b must be true of every node of the objects represented by the DESCRIBED-OBJ.

OBJECT-SCHEMA. An OBJECT-SCHEMA has the recursive unordered form,

a. <LOC-PROC> (OBJECT-SCHEMA)
b. ATTRIBUTE <word>

There may be any number of items of the form a or of the form b, including none. INITIAL-OBJ in Fig. 30 is an example of an OBJECT-SCHEMA.

DIFF-ORDERING. DIFF-ORDERING is a list of FEATUREs and groups of FEATUREs. The FEATUREs of the same group are enclosed in parentheses. (See Fig. 30.)

TABLE-OF-CONNECTIONS. TABLE-OF-CONNECTIONS is a list of items of the form

a. <FEATURE> (<list of operators>)

COMMON-DIFFERENCE stands for all FEATUREs. (See Fig. 30.)

COMPARE-OBJECTS. COMPARE-OBJECTS has the form

a. (BASIC-MATCH)

where BASIC-MATCH has the ordered form

b. COMP-FEAT-LIST <list of FEATUREs>
c. SUBEXPRESSIONS.

c is optional. In Fig. 30, c is missing which indicates that differences are only observed at the TOP-NODE of objects. The FEATUREs following COMP-FEAT-LIST are the type of differences that the match looks for.

Additional Facilities

Any TEST can be universally quantified over one variable. Such a TEST is treated as an arbitrary number of TESTs--one TEST for each value of the variable. The test is true, only if it is true for all values of the variable. Quantification is syntactically indicated by the occurrence, in a TEST, of FOR-ALL followed by a variable followed by an argument which designates a SET. In such a TEST, the variable is quantified over all members of the SET. (The Tower of Hanoi task specification uses this facility. See pages 201-2.)

A FEATURE can be used to designate a feature of any type of data structure and not only features of OBJECT-SCHEMAs. However, unless stated explicitly, FEATUREs refer to the implicit object. Whenever PARTICULAR, followed by the name of a data structure, occurs in a FEATURE, the FEATURE

refers to that data structure. (An example occurs in the Tower of Hanoi task specification (See page 202.)

The syntax of OBJECT-SCHEMA in the external language is clumsy for expressing the objects of some tasks. For example,

$$\int tdt$$

expressed as an OBJECT-SCHEMA is

```
(SYMBOL INTEGRAL LEFT (SYMBOL * LEFT (SYMBOL T)
 RIGHT ( SYMBOL D LEFT (SYMBOL T ))))
```

To alleviate this clumsiness, there are some conversion routines which allow OBJECT-SCHEMA to be expressed in a language with a different syntax.

In the LIST mode of translation, a string of symbols enclosed by matching parentheses are loaded as a data structure. In this mode, the translator does not consider (and) as meta-words even though it does match the (to). Objects and operators whose statements contain parentheses can be translated in the LIST mode and converted to internal form after translation. For example, the operators and initial situation of integration (Fig. 31) are translated in the LIST mode.

Example

The translation of the missionaries and cannibals task in Fig. 30 illustrates the way in which the text is processed. Starting at the top of Fig. 30, RENAME designates the mode of translation and the translator assigns LEFT and RIGHT to be the new names of FIRST and SECOND, respectively.) signifies the end of the scope of RENAME, and DECLARE changes the mode of translation. BOAT is then assigned the type ATTRIBUTE and the other declaration statements are processed, similarly. The next) indicates the end of the scope of DECLARE and TASK-STRUCTURES becomes the mode of translation.

TOP-GOAL, whose form is identical in all tasks, is the first data structure to be translated. TOP-GOAL is defined by the text, encosed in parentheses, which follows it. The first) following TOP-GOAL signifies the end of TOP-GOAL. Since the next word is INITIAL-OBJ the translator considers it to be the name of the next data structure to be translated. INITIAL-OBJ does not correspond to an IPL symbol in the basic vocabulary of GPS and is assigned a new IPL symbol for this task. (The same IPL symbol is also used for the occurrence of INITIAL-OBJ in TOP-GOAL.) INITIAL-OBJ is translated according to the form of an OBJECT-SCHEMA because it has been so declared. The) which matches the (following INITIAL-OBJ indicates the end of its definiton.

After translating DESCRIBED-OBJ in the same way, the EXPRES, X+Y is translated. X+Y and the three SETs which follow are used in the definition of the M-C-OPR.

FROM-SIDE-TESTS is translated as V-TESTS whose form is a disjunctive set of TESTs. FROM-SIDE-TESTs consists of the two TESTs, enclosed in the parentheses that follow it. TO-SIDE-TESTs are translated similarly. Both FROM-SIDE-TESTs and TO-SIDE-TESTs are used in the definition of the M-C-OPR.

The M-C-OPR, which is translated as a MOVE-OPERATOR, consists of the text enclosed in the parentheses that follows it. VAR-DOMAIN indicates that the four TESTs which follow specify the legitimate values of the variables in the M-C-OPR. MOVES indicates the end of the scope of VAR-DOMAIN and that the TRANSFORMATIONs of the operator follow. POST-TESTs signifies that the third TRANSFORMATION is the last and that the following constraints must be satisfied by any object produced by the M-C-OPR in order for it to be feasible. The POST-TESTS consist of two disjunctive sets of TESTs, which is indicated by the occurrences of TRUE.

After translating the M-C-OPR the types of differences B-L through M-R are translated. Each of these data structures are translated as FEATUREs. They are used during problem solving as difference types rather than the arguments of TESTs only because their names occur in DIFF-ORDERING and BASIC-MATCH instead of in TESTs.

DIFF-ORDERING divides the types of differences into two groups (each enclosed in parentheses). The types of differences in the first group are considered more difficult than those in the second group.

TABLE-OF-CONNECTIONs consists of a single type of difference (COMMON-DIFFERENCE, which stands for any type of difference) with which a list of one operator is associated.

After translating the next four data structures, the translator notices that the) matches the (following TASK-STRUCTURES, which indicates the end of its scope. The meta-word END terminates translation and control is transferred to the PROBLEM-SOLVING-EXECUTIVE of GPS.

CHAPTER V: REPRESENTATION AND GENERALITY

The interdependence of the power and generality of a problem solver was discussed in Chapter II. In the construction of a general problem solver, employing a fixed set of problem solving techniques, the internal representation is critical: it must be general so that tasks can be expressed in it; yet its structure must be simple enough for the problem solving techniques to be applicable. Since the techniques require that certain information be abstracted from the internal representation, they are applicable only if processes which abstract the necessary information from the internal representation are feasible. Thus, the difficulty of constructing a general problem solver is determined primarily by the variety and complexity of its problem solving techniques.

Chapter III described the problem solving techniques of GPS and Chapter IV described the internal representation of GPS. Thus, this chapter can now discuss the impact that the problem solving techniques had on the way in which the internal representation of GPS-2-5 was generalized. To simplify the following discussion, the demands of the problem solving techniques of GPS are summarized in Fig. 36; this is a list of processes that GPS must perform regardless what internal representation is used. These processes are given in a simplified form. For example, in some cases objects are not represented individually, but are represented as classes of objects. In such cases, object-comparison (Fig. 36) must test if two classes of object have any common members.

The first section of this chapter describes the properties of several different modes of representation and illustrates their correspondence to the types of objects and operators described in Chapter IV. The remainder of the chapter can then illustrate the interaction between the problem

1. Object-comparison--test if two objects represent the same situation, or if two classes of objects have any common members.

2. Object-difference--find a difference between two objects if they do not represent the same situation.

3. Operator-application--produce the result of applying an operator to an object if the operator is feasible.

4. Operator-difference--find a difference between an object and the class of objects to which an operator can be applied if the operator is not applicable to the object.

5. Desirability-selection--select from the set of all operators those operators which are desirable.

6. Feasibility-selection--select from a set of operators those operators which are feasible.

7. Canonization--find the canonical name of an uncanonized data structure (might depend on type of structure).

FIGURE 36. Processes required by the problem solving methods of GPS.

solving techniques of GPS and its internal representation. Each section discusses some aspect of proposed tasks that prevented them from being expressed in the internal representation of GPS-2-5 (the version available at the initiation of this work). For each, the attempts to alleviate the deficiency in the internal representation are related; some of these attempts were successful; others could not be adequately carried out within the existing program.

A. MODES OF REPRESENTATION

Before describing the different modes of representation used in GPS, we need a framework into which each can be cast. Although not all modes of representation are describable within the framework introduced below, it is adequate for the types of representation that are used in GPS. After describing the different modes of representation of objects, we will generalize them to include the representation of operators.

Framework for the Representation of Objects

The representation of an object is given by a set of information-units. Each information-unit has the same generic form: A Boolean-function of several arguments (in most cases two). The arguments can be atomic-constants--either symbolic or numeric. An argument can also be a feature of an object, which is a function whose domain is objects and whose range is values. A FEATURE is a special kind of feature. An example of an information-unit is

> "the number of missionaries at the left bank
> of the river equals 3."

In this information unit

> the Boolean function is "equals";
> the feature is "the number of missionaries at the left";
> the atomic-constant is "3".

This information-unit represents the fact that there are three missionaries at the left bank of the river in the object.

A subexpression is a subset of the information-units which represent the total object. A subexpression-name is a function whose domain is objects and whose range is subexpressions. An example of a subexpression-name is, "left" and its value in the initial situation of the missionaries and cannibals task is the subexpression consisting of the following three information-units:

```
missionaries at left = 3;
cannibals at left = 3;
boat at left = yes.
```

Models

One mode of representation, which will be called the model, is a set of information-units all of which have the same form:

```
Boolean function is =
first argument is a feature
second argument is an atomic-constant.
```

The set of information-units that comprise a model must be consistent--no two information-units contain the same feature; and complete--each feature is contained in one information unit.

For example, a model of the object,

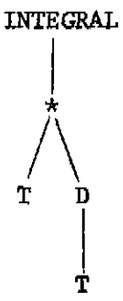
$$\int t\, dt, \qquad (1)$$

is given in Fig. 37.a, assuming that t is not a free variable and that t and dt are ordered. Fig. 37.a uses the terminology of the formulation of integration in Fig. 31. In this example, FEATUREs denote features. Many features do not appear in Fig. 37.a; e.g.,

LEFT-LEFT-LEFT SYMBOL.

(a) SYMBOL = INTEGRAL
LEFT SYMBOL = *
LEFT-LEFT-SYMBOL = T
LEFT-RIGHT-SYMBOL = D
LEFT-RIGHT-LEFT-SYMBOL = T

(b)



(c) α SYMBOL = T
β SYMBOL = D
γ SYMBOL = T
SYMBOL = INTEGRAL
LEFT SYMBOL = *

(d) (TEX-DESCRIPTION

1. THE SYMBOL EQUALS INTEGRAL .
2. THE LEFT SYMBOL EQUALS * .
3. THE LEFT-RIGHT SYMBOL EQUALS D .
4. LEFT-LEFT EQUALS LEFT-RIGHT-LEFT .)

FIGURE 37. Different representations of the integral, $\int$ t dt : (a) is a model; (b) is an OBJECT-SCHEMA tree structure; (c) is an unordered-schema; (d) is a DESCRIBED-OBJ.

Nevertheless, Fig. 37.a is "complete" because, by convention, all of those features that do not appear explicitly equal the special atomic-constant, UNDEFINED.

The key property of models is that their identity test is a match provided that the models use the same set of features. That is, two models are identical if and only if the atomic-constants in each pair of information-units which contain the same features are identical. The match must place information-units with the same features into correspondence which requires search if the information-units do not occur in some canonical order. But there is one and only one correspondence.

Schemas

There are two different kinds of schemas which, for the purpose of this chapter, will be considered as a single mode of representation. The simplest kind of schema has the same definition as a model with the exception that the atomic-constant in an information-unit may be (but not necessarily is) replaced by a variable whose domain is atomic-constants. We will not distinguish between those variables that are universally quantified and those variables whose domain is restricted, even though any restriction on the domain of variables must be given either explicitly in the encoding of the objects or by some uniform convention. The interpretation of schemas is that they represent a class of models each of which can be obtained by the substitution for variables. Hence, a model is the special kind of schema in which no variables appear.

The second kind of schema is the same as the one described above with the exception that some of the information-units may have the form·

```
Boolean function is equals;
first argument is a subexpression-name;
second argument is a variable whose domain
  is subexpressions.
```

For example, Fig. 37.a is a schema representation of (1), assuming t and dt are ordered. OBJECT-SCHEMAS are schemas and if an OBJECT-SCHEMA contains no variables, it is a model as well as a schema.

Fig. 37.b is the tree structure representation of the OBJECT-SCHEMA represented as a schema in Fig. 37.a. Any variables in OBJECT-SCHEMAs stand for subexpressions. Hence, if T is a free variable, Fig. 37.b represents the class of models whose LEFT-LEFT and LEFT-RIGHT-LEFT subexpressions can be any subexpression.

The key property of a schema is similar to that of a model: The identity test for two schemas is a match that incorporates the substitution for variables.

Unordered-schemas

The unordered-schema mode of representation is the generalization of schemas in which the features or subexpression-names in the information-units may (or may not) be replaced by variables whose domains are features or subexpression-names, respectively. Fig. 37.c is (1) represented as an unordered-schema, assuming that t and dt are unordered. α, β, γ are variable features. If α is LEFT-LEFT SYMBOL, then β must be LEFT-RIGHT SYMBOL and γ must be LEFT-RIGHT-LEFT SYMBOL. On the other hand, if α is LEFT-RIGHT SYMBOL, then β must be LEFT-LEFT SYMBOL and γ must be LEFT-LEFT-LEFT SYMBOL. The variable features in Fig. 37.c are not independent and are not universally quantified but nevertheless are variables.

Considerable complexity is introduced by using variable features and subexpression-names because there is no unique way to place information-units in correspondence as there was in schemas. However, the key property of unordered-schemas is that the identity test is still a match which incorporates substitution of variables, provided that it substitutes values for variable features.

Characteristic-lists

In the modes or representation described so far, the only Boolean function used is equal. The match can be used for an identity test only because equality is both transitive and reflexive. That is, two information-units which contain the same feature are identical only if their atomic constants are identical. When Boolean functions that are intransitive or irreflexive are used, the identity test is more complicated. For example, the two information-units

a. α is not less than β
b. $\alpha \neq \beta$

is equivalent to the single information-unit

$\alpha > \beta$

(α is a feature and β is a number).

A characteristic-list, a mode of representation, consists of a set of information-units, each of which has the generic form

$R(\alpha,\beta)$

where

a. $R \in \Re$ a set of Boolean functions;
b. α is a feature or a subexpression-name;
c. β is a feature, subexpression-name, subexpression or atomic-constant.

A characteristic-list is not necessarily complete in the sense that a model must be complete. If none of the information-units in a characteristic-list contain a particular feature, by convention its value is irrelevant whereas in a model its value was assumed to be UNDEFINED. Two information-units may contain the same feature and still be consistant because some of the Boolean functions may be intransitive or irreflexive and because an information-unit

may be a Boolean function of two features. For example, the two information-units

a. $\alpha > \beta$
b. $\beta < 5$

where α and β are features are certainly not inconsistant.

DESCRIBED-OBJs in GPS are characteristic-lists. Fig. 37.d is a DESCRIBED-OBJ which represents (1).

Characteristic-lists have the interesting property that all schemas can be represented as characteristic-lists even though the latter contain no explicit variables. (Since unmentioned features in characteristic-lists can have any value, they have the flavor of a set of independent, universally quantified variables.) We will not prove that characteristic-lists are as general as schemas. Instead this fact is demonstrated by expressing in Appendix B the operators of the logic task[1] as both FORM-OPERATORs (which are schemas) and MOVE-OPERATORs without variables (which are characteristic-lists). Note that some of the MOVE-OPERATORs in Appendix B can only be expressed as two FORM-OPERATORs.

Characteristic-list-schemas

The characteristic-list-schema mode of representation is a generalization of the characteristic-list which is obtained by allowing the use of variables for either argument of information-units. Thus, characteristic-lists are those characteristic-lists-schema that do not contain variables.

[1]Experiments with previous versions of GPS used, extensively, the formulation of the task of proving a theorem in the propositional calculus which is described on page 30. These are the operators from this specification of logic.

None of the data structures which represent objects in GPS are characteristic-list-schemas which contain variables. DESCRIBED-OBJs do not contain any explicit variables, except for those variables associated with FOR-ALL. However, there are operators, i.e., MOVE-OPERATORs, that contain variables and hence are characteristic-list-schemas.

Test-process

One way to represent an object is as a program which takes a single object as its input and provides 'yes' or 'no' as its output. Such a program represents the object, A if its execution, when A is the input, results in 'yes' and if its execution on all other inputs results in 'no'. We will call such a program a test-process. A test-process can also be used to represent a class of objects. In this case, the result of the program will be 'yes' whenever its input is a member of the class and 'no' for all other inputs. The desired situation in checkers is represented as a test-process in Samuel [53].

A test-process is the mode of representation consisting of a single information unit of the form

R (total object).

R is the Boolean function defined by the test-process program. Total object is the subexpression-name that designates the entire object. For example, a test-process representation of (1) is a program which checks if the input contains the information-units in Fig. 37.a. This implies that the test-process must understand how information-units are encoded in its input and that the input is expressed in a different mode of representation.

One advantage of a test-process is its generality. Any object represented in one of the other modes of representation can be represented as a test-process if the test-process is expressed in a general programming

language, e.g., IPL. Another advantage of a test-process is that its identity test is the test-process itself. The main disadvantage of test-processes is that they are unit processes as far as GPS is concerned. It is not feasible for GPS to analyze the program structure of a test-process to determine its semantics.

Modes of Representation of Operators

In the following, we will only consider those operators which have one object (or one class of objects) as an input and one object (or one class of objects) as an output. This case can be generalized easily to the several input, several output case. Processing considerations such as operator-application (Fig. 36) will not be dealt with in this section.

Operators involve statements about two objects: one is the input object, the other is the output object. This implies the existence of some device--semantic or syntactic--that permits reference to either. For the purpose of this discussion, features and subexpression-names preceded by an asterisk will refer to the output object; other features and subexpression-names will refer to the input object. (This syntactic device is not used in GPS.) The modes or representation for operators are obtained from the modes of representation of objects by allowing any of the features or subexpression-names to be preceded by an asterisk.

FORM-OPERATORs are the union of the two sets of information-units which represent the input and output schemas, respectively. The features and subexpression-name in the latter are preceded by asterisks. Fig. 38.a is the operator schema of the FORM-OPERATOR in Fig. 38.b.

The TRANSFORMATIONs of MOVE-OPERATORs are basically characteristic-list-schemas. For example, TRANSFORMATIONs of the form,

MOVE F1 TO F2,

(a) SYMBOL = ∫
LEFT SYMBOL = D
LEFT-LEFT SYMBOL = U
*TOP-NODE = U

(b) Input: ∫ Output: U
|
D
|
U

(c) sum (M FROM-SIDE ,-X, *M *FROM-SIDE)
sum (M TO-SIDE , X , *M *TO-SIDE)
M FROM-SIDE NOT-LESS-THAN X

(d) (MOVES
DECREASE BY THE AMOUNT X THE M AT THE FROM-SIDE
AND ADD IT TO THE M AT THE FROM-SIDE .
POST-TESTS
M AT THE LEFT IS GREATER THAN THE M AT THE TO-SIDE .)

FIGURE 38: (a) is a schema encoded as a tree structure in (b).
(c) is a characteristic-list-schema encoded as a MOVE-OPERATOR in (d).

where F1 and F2 are features, are equivalent to the three information-units which incorporate the asterisk convention:

DEFINED (F1)
EQUALS (F1, *F2)
UNDEFINED (*F1).

Since all TRANSFORMATIONs can be restated as several information-units, MOVE-OPERATORs are characteristic-list-schemas. Fig. 38.c is the set of information-units of the MOVE-OPERATOR in Fig. 38.d. In Fig. 38.c, 'sum' is a Boolean function which is true if the sum of the first two arguments equals the third.

The modes of representation introduced in this section are summarized in Fig. 39. The remainder of this chapter discusses inadequacies of the modes of representation used in GPS-2-5 and the extent to which these inadequacies were alleviated by the introduction of new modes of representation in GPS. Fig. 40 summarizes the role of the modes of representation in both.

B. DESIRED SITUATION

In many tasks, the desired situation is not a single object, but a large class of objects. In GPS-2-5, schemas and lists of schemas were the only means of expressing classes of objects, and they are not an adequate representation for many classes of objects. In integration, for example, the desired situation is an expression that does not contain an '$\int$' and there are an infinite number of such expressions. A schema cannot represent this class of objects because a schema implies identity of form. Similarly, no finite list of schemas is adequate.

A test-process can be used to represent the desired situation in integration. An example would be a program that generates the symbols of an object and tests if any of them is '$\int$'; if none are '$\int$', the program

1. Model--a complete and consistent set of information-units of the form--feature equals atomic-constant. A match is an identity test for two models.

2. Schema--a model in which the atomic-constants may be replaced by variables and some of the information-units may be of the form--subexpression-name equals variable. A match with substitution for variables is an identity test for two schemas.

3. Unordered-schema--a schema in which the features and subexpression-names may be replaced by variables. A match which pairs elements and substitutes for variables is an identity-test for two unordered-schemas.

4. Characteristic-list--a set of information-units of the form--Boolean function (α,β)--where α is a feature or subexpression-name and β is a feature, subexpression-name, subexpression, or atomic-constant. An identity test for two characteristic-lists is more complex than a match.

5. Characteristic-list-schema--a characteristic-list in which features, subexpression-names, subexpressions and atomic-constants may be replaced by variables. An identity test for two characteristic-list-schemas is more complex than a match.

6. Test-process--a complex Boolean function defined in a programming language. A test-process is its own identity test.

FIGURE 39. A summary of several modes of representation.

(a) A summary of the representation of GPS-2-5:

1. Initial situation: schema; list of schemas.
2. Desired situation: schema.
3. Operators: schemas.

(b) A summary of the internal representation of the current version of GPS:

1. Initial situation: schema; list of schemas; (unordered-schema in the integration task).
2. Desired situation: schema; characteristic-list.
3. Operators: schemas; characteristic-list-schemas.

(c) A list of the types of objects and operators in GPS together with their mode of representation:

1. OBJECT-SCHEMA: schema; (unordered-schema in the integration task).
2. DESCRIBED: characteristic-list.
3. FORM-OPERATOR: schema.
4. MOVE-OPERATOR: characteristic-list-schema.

FIGURE 40. A summary of the representation of both GPS-2-5 and the current version of GPS.

signals that the object is a member of the class of objects not requiring integration.

If we consider how to accomplish the requirements in Fig. 36 when the desired situation is represented as a test-process, we find that some are easy--object-comparison[2], which is the test-process itself--while others are formidable--object-difference. Object-difference processes could be imbedded in the test-processes, if their execution would produce a difference whenever the input object was not represented by the test-process. This solution would place a large burden on the person who constructed the test-process program. The only other way to obtain differences is to analyze the test-process program to discover the conditions that give rise to the negative signal. However, such an analysis requires an understanding of the programming language and is not feasible.

A characteristic-list--in particular, a DESCRIBED-OBJ--can be used to represent the desired situation in integration. In GPS the desired situation can be represented as a DESCRIBED-OBJ. However, a DESCRIBED-OBJ cannot be used to represent the initial situation or objects derived from it since GPS cannot apply an operator to a DESCRIBED-OBJ (operator-application) and cannot produce a DESCRIBED-OBJ as the result of an application of an operator. Thus, for object-comparison and object-difference, GPS need only compare an OBJECT-SCHEMA and a DESCRIBED-OBJ because the necessity for comparing two DESCRIBED-OBJ never arises.

In comparing two data structures whose modes of representation are different, e.g., OBJECT-SCHEMA versus DESCRIBED-OBJ, a match cannot be used

---

2Throughout this chapter are used the short names of the demands of the problem solving techniques of GPS introduced in Fig. 36.

for the comparison process because there is no general way to place the two structures into correspondence. An OBJECT-SCHEMA and a DESCRIBED-OBJ are compared by an interpreter that understands the semantics of the RELATIONs and the format of a DESCRIBED-OBJ. The interpreter evaluates the arguments of the TESTs in the DESCRIBED-OBJ relative to the OBJECT-SCHEMA and if it finds all of the TESTs to be true, it signals that the OBJECT-SCHEMA and the DESCRIBED-OBJ represent the same situation. For example, the OBJECT-SCHEMA in Fig. 41.a and the DESCRIBED-OBJ in Fig. 41.b represent the same class of objects, if U is a variable, and the MATCH-DIFF-METHOD can recognize this fact. (In Fig. 41.a, INTEGRAL, D, and U are values of the ATTRIBUTE, SYMBOL, and the LOC-PROG, LEFT refer to the left-most branch of the TOP-NODE.) In comparing Fig. 41.a to Fig. 41.b, the MATCH-DIFF METHOD recognizes EQUALS and TEX-DESCRIPTION as words which it understands, and recognizes that

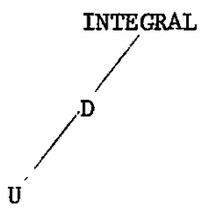
THE SYMBOL AT THE LEFT

(in Fig. 41.b) refers to the D in Fig. 41.a, etc.

When an OBJECT-SCHEMA and a DESCRIBED-OBJ do not represent the same situation, an object-difference process analyzes the TESTs in the DESCRIBED-OBJ which were not satisfied in order to produce a difference. Although the differences are implicit in the TESTs, it is important that they need not be a consideration in formulating the DESCRIBED-OBJ. For example, in matching OBJECT-SCHEMA in Fig. 41.a to the DESCRIBED-OBJ in Fig. 41.c, GPS would detect a difference because the TEST in Fig. 41.c is not true of the TOP-NODE. Only one difference would be detected because the TEST is true at the other two nodes. In analyzing this condition, GPS would detect the difference,

a. type of difference = the FEATURE, SYMBOL
b. location of difference = TOP-NODE
c. value of difference = a symbol other than INTEGRAL.

(a)



(b) (TEX-DESCRIPTION

1. THE SYMBOL EQUALS INTEGRAL .
2. THE SYMBOL AT THE LEFT EQUALS D .)

(c) (SUBEXPRESSION-TESTS

1. THE SYMBOL DOES NOT-EQUAL INTEGRAL .)

FIGURE 41. (a) The tree structure representation of an object in the integration task. (b) is the representation of (a) as a DESCRIBED-OBJ. (c) is the representation of the "desired situation" of the integration task as a DESCRIBED-OBJ. In this figure, U is a variable, SYMBOL is an ATTRIBUTE and LEFT is a LOC-PROG.

In general, GPS produces one difference for each TEST which is not satisfied. The type of the difference is a FEATURE that is an argument of the TEST; the location of the difference is the LOC-PROG which names the node to which the TEST was applied; and the value of the difference is the value which the FEATURE should have in order for the difference to be alleviated.

In order for GPS to determine the value of the difference, it must understand the RELATION of the TEST. For example, GPS must understand the semantics of NOT-EQUAL in order to determine the value of the difference in the previous example. GPS understands the semantics of the RELATIONs only because they are task invariant. On the other hand, GPS cannot understand the semantics of the types of differences, because they vary from task to task. GPS knows only that SYMBOL in the integration task is an ATTRIBUTE (this fact is given in the task description) and knows to process it in the same way that it processes, for example, BOAT in the missionaries and cannibals task.

Allowing the desired situation to be a DESCRIBED-OBJ does not affect the existing processes for operator-application, operator-difference, feasibility-selection, and desirability-selection because operators are never applied to DESCRIBED-OBJs. A DESCRIBED-OBJ never needs to be canonized because new ones are not generated during problem solving. However, in any difference detected between an OBJECT-SCHEMA and a DESCRIBED-OBJ, the type of difference does not have a canonical name. Canonization of such a type of difference is accomplished by the same process which canonizes OBJECT-SCHEMAs because they are all FEATUREs which can be matched by the IDENTITY-MATCH-METHOD. If the types of differences were more complex structures, such as IPL programs, the canonization process would necessarily be more complex.

C. OPERATORS

In GPS-2-5, all operators were expressed as schemas. This is a convenient representation for some operators, particularly those found in mathematical calculi. However, the operators of other tasks cannot be expressed, conveniently, if at all, as FORM-OPERATORs; e.g., the operators of the missionaries and cannibals task. To alleviate this difficulty, the internal representation of operators was extended to include MOVE-OPERATORs (characteristic-list-schemas).

The addition of MOVE-OPERATORs to the internal representation does not affect object-comparison or object-difference because they are concerned only with objects. However, the other existing processes listed in Fig. 36 were not adequate for MOVE-OPERATORs; the modification of each will be discussed below.

Operator-application

A new operator-application process had to be added so that a MOVE-OPERATOR could be applied to an OBJECT-SCHEMA. MOVE-OPERATORs always have one OBJECT-SCHEMA as input and produce, if successful, a single OBJECT-SCHEMA as the result. No attempt was made to apply a MOVE-OPERATOR to a DESCRIBED-OBJ.

The operator-application process for MOVE-OPERATORs tests an operator's feasibility (PRETESTS are satisfied, etc.) and, if feasible, produces the result. This process interprets the semantics of the TRANSFORMATIONs which involves understanding the semantics of the OPERATIONs. But GPS does not understand the semantics of task dependent symbols; e.g., all ATTRIBUTEs are processed similarly. The operator-application process for MOVE-OPERATOR also interprets the semantics of the TESTs in a MOVE-OPERATOR, which is

accomplished by the same apparatus used in the object-comparison process for an OBJECT-SCHEMA and a DESCRIBED-OBJ.

Operator-difference

The operator-difference process for a MOVE-OPERATOR is quite similar to the object-difference process for an OBJECT-SCHEMA and a DESCRIBED-OBJ. In fact, when one of the TESTs in PRETESTs is not satisfied, the two processes are identical.

The failure of a TEST in the set of TESTs in VAR-DOMAIN causes the operator to fail unconditionally and no difference is produced. This condition indicates that one of the variables has an illegitimate value and thus, the operator is infeasible, independent of the object to which it is applied.

The inapplicability of a TRANSFORMATION will result in a difference. To produce such a difference GPS must understand the semantics of the OPERATIONs. For example, the TRANSFORMATION,

MOVE THE BOAT AT THE FROM-SIDE TO THE BOAT AT TO-SIDE

with FROM-SIDE equal to LEFT and TO-SIDE equal to RIGHT, is not applicable to an OBJECT-SCHEMA in which the BOAT is at the RIGHT. An attempt to apply this TRANSFORMATION will result in the type of difference,

LEFT BOAT,

because MOVE requires the first argument of the TRANSFORMATION to be DEFINED. This type of difference is obtained by substituting LEFT for FROM-SIDE in the first argument of the TRANSFORMATION.

This example illustrates the important fact that types of differences are not atomic symbols, but rather, data structures (FEATUREs), whose generic form is known by GPS. Since some of the FEATURES in the TRANSFORMA-

TIONs of MOVE-OPERATORs contain variables, it is not possible to preassign each individual type of difference to be an atomic symbol (as was true in GPS-2-5). In a particular application, such variables will have values; a FEATURE which contains specified variables may be produced as a type of difference. If the variables were specified differently, the FEATURE would be equivalent to another type of difference. For example, the application of the TRANSFORMATION above with FROM-SIDE equal to RIGHT and TO-SIDE equal to LEFT, might result in the difference,

RIGHT BOAT.

In the previous example, the same argument of the same TRANSFORMATION resulted in a different type of difference because the value of FROM-SIDE was different.

The current operator-difference process does not produce a difference upon the failure of a TEST in POST-TESTs. Instead, this condition results in unconditional infeasibility. POST-TESTS are stated relative to the resultant object instead of the input object. Hence, the FEATURE that is incorrect and causes the failure probably was modified in applying the TRANSFORMATIONs. The current process for producing a difference is not sophisticated enough to state the difference relative to the input object. Such a restatement is required because GPS assumes that all differences are stated relative to the input object.

Desirability-selection

The REDUCE-METHOD does not select operators randomly but selects desirable operators--those which modify in a desirable way that feature of an object to which the difference of the REDUCE goal pertains. TABLE-OF-CONNECTIONS is a means of providing information about the desirability of operators. Associated with each type of difference is a list of the operators

that have the capability of reducing the type of difference. Although the information in TABLE-OF-CONNECTIONS should be discovered by GPS; normally, GPS-2-5 was given the TABLE-OF-CONNECTIONS exogenously. However, one attempt was made to construct the TABLE-OF-CONNECTIONS for a set of FORM-OPERATORS (Newell [36]). To determine the differences for which a FORM-OPERATOR was relevant, its input form was matched to its output form. This was possible, since both were OBJECT-SCHEMAs. The differences that were produced as a result of the match were those which the FORM-OPERATOR had the capability of reducing.

For example, one of the FORM-OPERATORs used in the logic task is

(A⊃B) YIELDS (¬AVB)

where A and B are variables. Matching the left hand expression to the right hand expression results in the types of differences:

a. logical connective
b. sign of the first operand

Thus, this operator was considered desirable to changing either the logical connective or the sign of the first operand[3]. This example is interesting because it demonstrates the dependence of processing on the representation. For example, it is not feasible to construct the TABLE-OF-CONNECTIONS if the operators are represented as test-processes.

Although the TABLE-OF-CONNECTIONS is adequate for expressing the desirability of schema operator, it is not adequate for characteristic-list-

[3]The actual process described in Newell [36] was somewhat more complex but the basic idea is the same. A method in a predecessor of GPS-2-5 used a similar process, dynamically, to determine desirability.

schemas. Even after selection, characteristic-list-schemas operators may contain variables whose values may determine the desirability of the operator. In the extreme case, illustrated by missionaries and cannibals, only one operator appears in the TABLE-OF-CONNECTIONS, which results in no selectivity at all. Since MOVE-OPERATORs are not schemas, the technique used to discover the desirability of FORM-OPERATORs in GPS-2-5 is not applicable. Thus, a new desirability-selection mechanism had to be added to GPS for determining the desirability of MOVE-OPERATORs and the desirable values for pertinent variables in MOVE-OPERATORs.

The desirability-selection process for MOVE-OPERATORs can best be described by an example. Suppose that in attempting the missionaries and cannibals task, an operator was wanted for reducing the difference,

the BOAT at the LEFT is UNDEFINED.

The M-C-OPR would be selected from the TABLE-OF-CONNECTIONS (see Fig. 30). The desirability-selection process attempts to make the functions of the TRANSFORMATIONs desirable by substitution for variables. The functions of the second and third TRANSFORMATIONs are not relevant to reducing the difference because they do not affect the value of the BOAT at the LEFT. But the first TRANSFORMATION, since its OPERATION is MOVE, is desirable provided that its second argument is identical to the type of difference,

the BOAT at the LEFT.

The desirability of this TRANSFORMATION is determined by matching its second argument to the type of difference (both are FEATURES). Since they are identical when TO-SIDE equals LEFT, the desirability-selection process reports that the M-C-OPR with TO-SIDE equal to LEFT is a desirable operator.

Although there is a special match routine for FEATUREs in the desirability-selection process, ideally the MATCH-DIFF-METHOD would be used by the desirability-selection process to match two FEATUREs. However, a special match was used because of the following peculiarities: Variables are bound only if the two FEATUREs can be made identical, whereas, in matching two OBJECT-SCHEMAs, variables are bound regardless of whether they can be made identical. In addition, the match incorporated in desirability-selection only substitutes those values for the variables which satisfy VAR-DOMAIN.

The desirability-selection process depends on the homogeneous representation of types of differences and the arguments of TRANSFORMATIONs. In matching them, the task dependent detail of the FEATUREs cancels out. In the missionaries and cannibals task, for example, the match recognizes that two FEATUREs, both of which have BOAT and LEFT as their ATTRIBUTE and LOC-PROG, respectively, are identical. The match recognizes that the ATTRIBUTE in both FEATUREs is the same symbol and thus does not need to understand the semantics of BOAT. If their modes of representation were different, the comparison process would be more complicated than a match.

Desirability-selection is, in many respects, similar to operator-difference. In both cases, GPS must understand the semantics of the OPERATIONs which are the task invariant symbols in TRANSFORMATIONs. Note that GPS only understands the generic form of the arguments of the TRANSFORMATIONs, i.e., GPS knows that they are FEATUREs, CONSTANTs, etc., but does not attach any further significance to the task variant symbols in the TRANSFORMATIONs. A crucial property on which both desirability-selection and operator-difference depend is that FEATUREs are not atomic symbols but data structures in which variables may occur. (See pages

135-6.)

The use of variables in MOVE-OPERATORs allows many operators to be represented as a single data structure that is only marginally larger than the representation of any one of the operators. Not only does this make the representation more concise, but it allows many operators to be simultaneously analyzed in order to determine their desirability.

Feasibility-selection

Due to the way an operator is processed in GPS, it has been defined as a function represented as a single entity, whose domain and range are both sets of objects. A model of an operator is a degenerate case because both its domain and range are single objects. However, a model of an operator is still an operator according to the definition, because it is a single entity. Although models of operators are unparsimonious in both their memory and processing requirements, they will be used below to illustrate some properties of better modes of representation.

Schemas. A schema operator represents a class of model operators as a single entity. Before a schema operator can be applied to an object, the variables in the schema must be specified. Since a schema that contains no variables is a model, the specification of variables in a schema operator is equivalent to selecting a member from the class of model operators represented by the schema. Feasibility-selection requires that the model operator selected (variable specification) must be feasible if one exists. If an infeasible model operator were selected, GPS might still try to apply it by attempting to reduce a difference on the object. This would be a waste of effort if the schema contained a feasible model operator.

Feasibility-selection for FORM-OPERATORs is accomplished by matching the input form of the operator to the object to which the operator is being

applied and by making the necessary substitutions for variables (MATCH-DIFF-METHOD). However, the match can be used only because the modes of representation of the input form and the object are the same (both are always OBJECT-SCHEMAs).

Characteristic-lists. Characteristic-list operators like schema operators, represent a class of model operators as a single entity. In fact, every schema operator can be represented by a characteristic-list operator as demonstrated in APPENDIX B. Characteristic-list operators in GPS are MOVE-OPERATORs which do not contain variables. Since these operators are applied to OBJECT-SCHEMAs, the match cannot be used for feasibility-selection because their modes or representation are different. Moreover, since there are no variables in characteristic-lists, feasibility-selection for characteristic-list operators is of a different nature than that for schemas.

MOVE-OPERATORs describe how the output object of the operator differs from its input. By convention, any FEATUREs of the output object that are not specified explicitly by the TRANSFORMATIONs have the same values as their correspondents in the input object. Hence, the input object specifies degrees of freedom in a MOVE-OPERATOR just as the input object specifies degrees of freedom (the values of the variables) in a FORM-OPERATOR. Although the feasibility-selection (specification of degrees of freedom) for MOVE-OPERATORs that do not contain variables is free, there is a corresponding expense in that the operator-application process for MOVE-OPERATORs is considerably more complex than the substitution for variables in a form.

One advantage of MOVE-OPERATORs that do not contain variables is their ability to express classes of operators as a single entity which can only be expressed as several FORM-OPERATORs. An example is the MOVE-

OPERATOR, R1: a,b, in Appendix B[4]. In applying this MOVE-OPERATOR, the feasible model operator, if one exists, will be selected. But, if represented by the two corresponding FORM-OPERATORs, the REDUCE-METHOD must select one to be applied. If the wrong one is selected (i.e., the other one contains a feasible model operator) GPS may waste considerable effort by attempting to reduce differences between the object and the operator selected.

Characteristic-list-schemas. The use of variables in MOVE-OPERATORs allows many MOVE-OPERATORs which do not contain variables to be expressed as a single MOVE-OPERATOR. From the previous example, it would appear that the use of MOVE-OPERATORs that contain variables would lead to more efficient problem-solving because feasibility-selection can be applied to larger classes of model operators. Unfortunately, the current feasibility-selection process does not capitalize on the compactness of MOVE-OPERATORs that contain variables. GPS applies a MOVE-OPERATOR by first generating legal variable specifications (those for which the set of TESTs in VAR-DOMAIN of the MOVE-OPERATOR are satisfied) until it finds a set of values for which the operator is feasible or for which the difference between the operator and the object is not too difficult. This method for applying a MOVE-OPERATOR has two major disadvantages. First, the operator selected may not be the easiest operator. An infeasible model operator may be selected even though a different variable specification might have lead to the selection of a

4From this example, it might seem that the use of variables with restricted domains in FORM-OPERATORs would yield the generality of MOVE-OPERATORs. However, this generalization of FORM-OPERATORs would not help in the statement of the distributive law in Appendix B. That is, four of these generalized FORM-OPERATORs would be required to express R7.

feasible model operator. Since all variable specifications seem to be equally desirable (else further selections for desirability would have occurred), a better strategy would be to apply the easiest operator first.

The other disadvantage is that the number of different variable specifications may be large. Any specification of variables by the REDUCE-METHOD in order to insure the desirability of the operator considerably decreases the number of legitimate variable specifications that can be generated in attempting to apply the operator. In the tasks discussed in the next chapter, no case which had more than five different legitimate variable specifications was encountered and, in most cases, only one or two legitimate variable specifications could be generated. However, the number of variable specifications could conceivably be much larger for other tasks.

In the best of all worlds, the variable in a MOVE-OPERATOR would be assigned feasible values one at a time as they turned up in trying to apply the MOVE-OPERATOR. This is the main function of the match routine in applying a FORM-OPERATOR. But considerable complexity is introduced by unspecified variables in MOVE-OPERATORs. It is difficult to determine which values of the variables in MOVE-OPERATORs make the operator feasible. In the M-C-OPR in Fig. 30, for example, the POST-TESTS constrain the feasible value of X and Y even though X and Y do not appear explicitly in the POST-TESTs. In addition, TESTs, such as

THE M AT THE LEFT is NOT-LESS THAN THE C AT THE LEFT

constrain the feasible values of both X and Y simultaneously.

Another difficulty with specifying the variables in MOVE-OPERATORs as they turn up is the possibility of several feasible values. When variables do not have unique values that make the operator feasible, the application of

the operator will have different results. That is, the resultant class of objects cannot be represented by a single OBJECT-SCHEMA. In such cases, GPS must decide which member it will work with, since it can only do one thing at a time.

For example, in the missionaries and cannibals task when GPS attempts the goal of applying the M-C-OPR in Fig. 30 with TO-SIDE equal to LEFT, several different results can be produced depending on the values assigned to the variables, X and Y. When applied to the object,

(LEFT (M 1 C 1) RIGHT (M 2 C 2 BOAT YES))

the result can be either

(LEFT (M 3 C 1 BOAT YES) RIGHT (M 0 C 2))

or

(LEFT (M 2 C 2 BOAT YES) RIGHT (M 1 C 1)).

If the only reason for applying the operator was to move the BOAT to the LEFT bank of the river, both results would seem equally desirable, and X and Y would have two equally desirable and feasible values.

Canonization. MOVE-OPERATORS never need to be canonized because new MOVE-OPERATORs are never created. But, if desirability-selection specifies several variables in a MOVE-OPERATOR, then the goal of applying a MOVE-OPERATOR is created. A partially specified MOVE-OPERATOR is represented by a data structure consisting of a variable specification and the name of a MOVE-OPERATOR. Since new partially specified MOVE-OPERATORs are generated during problem solving, their canonization is required. All partially specified MOVE-OPERATORs are represented homogeneously, and the IDENTITY-MATCH-METHOD can match them by testing if the MOVE-OPERATOR and the values assigned to the variables are the same. Thus, the canonization

process of OBJECT-SCHEMAs can be used.

The only other data structures which must be canonized as a result of the addition of MOVE-OPERATORs are differences generated when a MOVE-OPERATOR is inapplicable. These differences are canonized in precisely the same way that the differences produced as a result of comparing an OBJECT-SCHEMA and a DESCRIBED-OBJ are canonized.

D. UNORDERED-SCHEMAS

An example of a task which can use unordered-schemas beneficially is integration. If multiplication and addition are represented as ordered binary functions, their associativity and commutativity must be expressed as operators. On the other hand, multiplication and addition can be represented as functions of an unordered set of arguments. Then the commutativity and associativity of multiplication and addition is expressed implicitly in the representation (provided that none of the arguments of multiplication was the multiplication of a set of arguments and none of the arguments of addition was the addition of a set of arguments). This representation has problem solving implications. For example, the

$$\int te^{t^2}\,dt$$

and

$$\int e^{t^2}\,t\,dt$$

would appear identical if the arguments of multiplication were unordered. But to make

$$\int t*(e^{t^2}*\,dt)$$

and

$$\int e^{t^2} * (t*dt)$$

identical would require the application of three operators.

To illustrate the problem solving efficiency that can be obtained by using unordered-schemas to represent unordered sets, consider the alternate representation of an unordered set: an ordered set, represented as a schema, and a permutation operator which permutes any two elements. To test the identity of two unordered sets represented as schemas, the permutation operator must be applied in an attempt to make the two schemas identical (to within a substitution for variables). Successive application of the permutation operator corresponds to searching the tree in Fig. 42. This search is unnecessary if unordered-schemas are used instead of schemas provided that the match routine can take into consideration their unordered property. Such a match routine might have to do some search in comparing two unordered-schemas. But if the match for unordered-schemas is only marginally more expensive than the match for schemas, the savings will be great because moving the permutation operator inside of the match compresses the tree defined by the initial situation and the operators.

Testing the identity of two unordered-schemas would be rather straight forward if they contained no variables. However, incorporating the substitution for variables in a process for matching two unordered-schemas introduce considerable complexity. To demonstrate this, consider matching the two unordered sets,

$$\{e^u,\ u,\ v\}$$

and

$$\{b, c, e^c\}$$

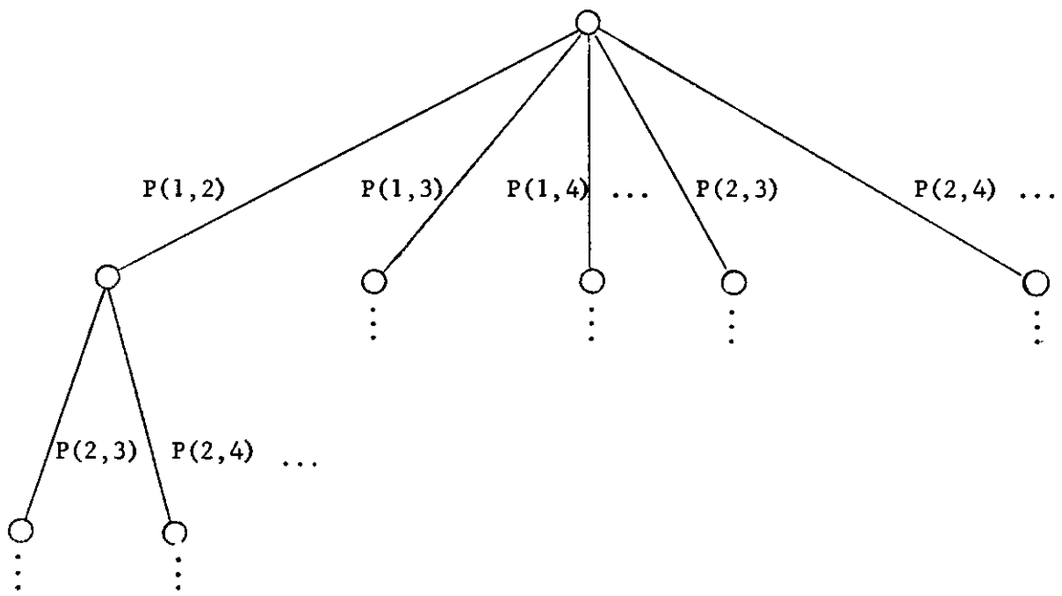


FIGURE 42. The tree defined by a set and a permutation operator, $P(\alpha,\beta)$, that permutes the elements, α and β.

in which u and v are variables[5]. Substituting b for u might seem desirable, since it makes a pair of elements of the sets identical. But this substitution would make the two sets different. The match must be sophisticated enough to substitute c for u (and b for v) in order to recognize that the two sets can be made identical.

Object-difference and operator-difference for unordered-schemas are difficult because there may be several pairings of the elements of the two sets which seem equally good. Consider the example of applying the operator,

$$\int e^u du = e^u \qquad (2)$$

in which u is a variable, to the object,

$$\int te^{t^2} dt \qquad (3)$$

The operator is applied by matching the left-hand side of the equation to the object and, if identical, using the right-hand side of the equation as the result. Obviously, the two sets of factors cannot be made identical because they have a different number of elements. But either the substitution of t for u or the substitution of t^2 for u will make one pair of elements identical. The latter is the correct substitution, but only because the operator,

$$t\ dt = 1/2\ d\ t^2$$

can be applied to (3) in order to make (2) applicable. This cannot be easily known at the time of matching the left-hand side of (2) to (3).

5This example also demonstrates that pairing elements lexicographically is not sufficient for expressions which contain variables.

Before GPS was given the integration task, the MATCH-DIFF-METHOD and the IDENTITY-MATCH-METHOD were modified so that they could match unordered sets. The modification consisted of several rules for pairing elements before matching them. First, an attempt is made to match elements that do not contain variables. An error in correspondence cannot be made pairing identical elements which do not contain variables, and such a pairing might prevent an unnecessary substitution for a variable. This modified match requires a list of properties which is used to rank elements according to their respective importance. This information, which is task dependent, is given to GPS exogenously as an IPL structure because it cannot be expressed in the external representation. After matching constant elements, other elements are selected in the order of their importance, to be matched allowing substitution for variables. However, if two elements in the same set are equally important (have the same property), these elements will be matched later. The strategy is to find a correspondence between unique, important elements of the two sets so that the substitution for variables is made correctly.

Finally, the remaining elements of the two sets are matched. The differences between the two sets are the unmatched elements. The match works strictly forward and cannot try two different substitutions for the same variable. Consequently, in some cases, the match will not recognize that two unordered-schemas can be made identical; but for the integration task, this match appears to be sufficient.

In applying the operator (2) to the object (3), for example, first the two exponential elements are matched because they are considered the most important. Upon substitution of t^2 for u they become identical. None of the other elements can be matched, and the difference reported is that the

number of elements in the set

{t, dt}

is too large (SET-SIZE).

Object-comparison, operator-application, feasibility-selection, and canonization for unordered-schemas are all satisfied by generalizing the match, because the match is incorporated in all of them. With the list of properties for ranking elements according to their importance, the generalized match can draw good correspondences between the elements of two sets even if the two sets are not identical. This together with the addition of a few set differences, (e.g., set is too small, set is too large) satisfies object-difference and operator-difference. The existing desirability-selection process is applicable for unordered-schema.

E. LARGE OBJECTS

The result of an application of an operator in GPS is always a new data structure, provided that it is not identical to some data structure created previously. GPS is only designed to handle simple problems (i.e., problems that require a limited amount of search) because it is not prepared to erase these data structures. Hence, the size of the data structures which represent objects is important; if too large, memory will be exhausted before even simple problems can be solved. For many tasks, this difficulty is critical because in GPS objects represent total situations and the total situations of some tasks are large. (There is no facility for processing object fragments as independent data structures.)

An example of a task in which the representation of objects is too

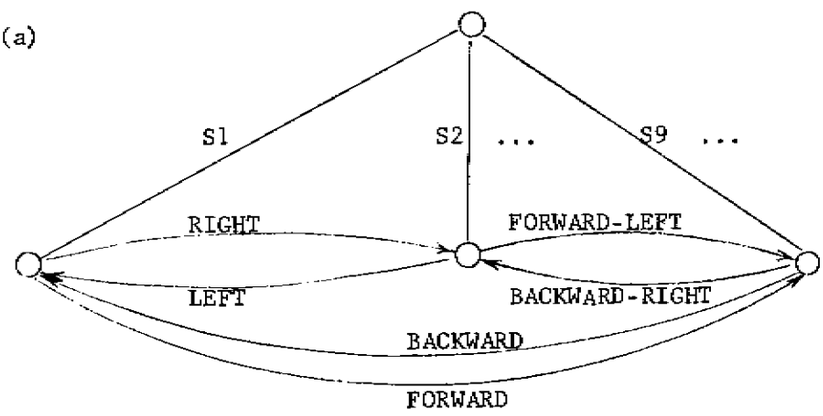
large is chess or any other task which uses a chess board[6], because each object must represent an entire board situation. By slightly generalizing the tree structure representation of OBJECT-SCHEMAs to allow any node to have more than one super node, a chess board could be represented as an OBJECT-SCHEMA. The TOP-NODE of an OBJECT-SCHEMA that represents a chess board has sixty-four branches; the nodes to which these branches lead represent the sixty-four squares of the chess board. For example, the branch labeled S1 in Fig. 43.a leads to the node which represents the square S1 in Fig. 43.b. Each square has eight branches, except for the border squares which have fewer than eight, leading from it to the eight neighboring squares. For example, the branch labeled FORWARD leading from the node, which represents S1, to the node, which represents S8, in Fig. 43.a, indicates that the square S8 is in the FORWARD direction from the square S1 in Fig. 43.b. The position of the pieces on the chess board would be given by the local description of the squares (nodes) along with any other information peculiar to a square, such as its color.

Although the representation in Fig. 43.a is convenient for processing[7], a single chess position is very large due to the elaborate system of branches between nodes. There are 484 branches between the nodes of Fig. 43.a. Since the internal representation of each branch requires at least two IPL-V words,

6An example of a non-game task that uses a chess board is the problem of placing eight queens on a chess board so that no queen can capture another queen.

7This representation is almost identical to the one used in Baylor [2].

(a)



(b)

FIGURE 43. (a) is the OBJECT-SCHEMA that represents the chess board whose squares are named in (b).

the internal representation of a single chess position would require more than 1,000 IPL words.

Obviously, the information represented by the system of branches in Fig. 43.a must be expressed to the problem solver somewhere in the specification of the task of playing chess. But since this information does not vary from chess position to chess position, it should be represented only once and not in the representation of every chess position generated during problem solving. One way to avoid this duplication of information is to represent chess positions independently of the chess board; e.g., as a list of pairs in which one member of a pair is a square and the other member is a piece on the square. This representation contains no information about which pieces are on adjacent squares, etc. Before considering a particular position, GPS would put the pieces on a board such as the one in Fig. 43.a and, after it had finished processing the position, it would remove the pieces from the board. This scheme would require the addition of a mechanism for "setting up" and "tearing down" chess positions.

An alternate representation for chess positions is to express the relationships between the squares of the chess board implicitly in the names of the squares instead of explicitly in the structure of each object. For example, the name S1 in Fig. 43.a could contain the information that the square at the RIGHT was S2 and the square in the FORWARD direction was S8, etc. That is, S1 would be a data structure as well as the name of a square and information peculiar to S1, e.g., S2 is at the RIGHT of S1, would be encoded in the data structure, S1, in some convenient form. This representation cannot be used in the current version of GPS because LOC-PROGs must be built into the structure of all objects. Since RIGHT, FORWARD, etc., are LOC-PROG--they correspond to the branches of tree structures--they must be in the

structure of objects regardless if the relationships between squares are expressed in the names of squares.

F. DIFFERENCES

In GPS, all types of differences are FEATURES and a particular difference indicates that the value of the FEATURE is incorrect. If the value of the FEATURE is a number, the difference designates the amount by which the number should be increased or decreased in order to alleviate the difference; if the value of the FEATURE is a symbolic constant, the difference designates the correct value of the FEATURE. These elementary differences are not an adequate representation for the differences of some tasks. That is, for some tasks, GPS's problem solving would appear aimless because these elementary differences are not sufficient to guide the search.

We can illustrate the inadequacies of the types of differences that are represented as FEATUREs by a detailed example of how GPS should solve simultaneous equations. Fig. 44 is an informal formulation for GPS of the task of solving two simultaneous equations. The initial situation is the set of the two OBJECT-SCHEMAs in Fig. 44.a. The desired situation expressed as the DESCRIBED-OBJ in Fig. 44.c represents an expression of the form--Y equals an expression which does not contain a Z. (Y and Z are considered the independent variables of the equations.)

The operators, given in Fig. 44.b can all be represented as FORM-OPERATORs. The $\longleftrightarrow$ indicates that the form on either side can be used as the input form and $\rightarrow$ indicates that the form to the left is the input form. We will assume that GPS can do algebraic simplification implicitly and that it understands that multiplication and addition are commutative and associative.

(a) Initial Situation:

1. A * Y + B * Z = Y + E
2. C * Z + Y = D * Z

(b) Operators:

1. (U + V = W)⟷ (U = W - V)
2. (U - V = W)⟷ (U = W + V)
3. (U * V = W)⟷ (U = W / V)
4. (U / V = W)⟷ (U = W * V)
5. (U = V) → (U - V = 0)
6. (U , U = V) → V
7. (U * (V + W))⟷ (U * V + U * W)

(c) Desired situation:

(TEX-DESCRIPTION

1. THE SYMBOL AT THE LEFT EQUALS Y .
2. THE SYMBOL EQUALS = .
3. THE SYMBOL AT X DOES NOT-EQUAL Y , FOR-ALL X IN THE RIGHT-SUBEXPRESSION .

SUBEXPRESSION-TESTS

THE SYMBOL DOES NOT-EQUAL Z .)

FIGURE 44. An informal formulation of solving two simultaneous equations.

Thus, GPS does not need to create GOALs to simplify expressions or to commute two factors.

Since the initial situation is a set of objects, GPS would select one and attempt to transform it into the desired situation. Suppose that the first equation in Fig. 44.a were selected. The current version of GPS would detect that the Y's and the Z in the object should be replaced by different SYMBOLs, e.g., one of the differences detected would be

RIGHT SYMBOL should not be a Y.

According to such differences, all of the operators would appear desirable, because they all replace SYMBOLs with other SYMBOLs. Thus, for this task, GPS must detect types of differences which are more global than the elementary types of differences which can be represented as FEATUREs. Suppose that GPS could detect differences that summarize what is wrong with the entire object or a subexpression of the object. GPS would recognize the difference that the first equation in Fig. 44.a contained one Z instead of none. The only operator capable of reducing this difference is the sixth operator in Fig. 44.b and V must stand for a subexpression which does not contain a Z if the operator is to reduce the difference.

GPS would attempt to apply this operator by setting U equal to Z in order to supply the first input and would look for the second input to the operator--an OBJECT-SCHEMA of the form

Z = V,

where V stands for some expression which does not contain a Z. To modify the second equation in Fig. 44.a so that it could be used as the second input to the operator, GPS would attempt to reduce the difference that the expression to the right of '=' contains a Z.

The first six operators in Fig. 44.b are all relevant to reducing the difference but only the fifth is applicable and GPS would produce the new object

$$(C * Z) + Y - (D * Z) = 0.$$

Since this object has two Zs to the left of '=' instead of one, the seventh operator would be applied to it to produce

$$Z * (C - D) + Y = 0,$$

which is still not of the form

$$Z = V.$$

The difference that the subexpression to the left of '=' is too large is not an adequate summary of this state of affairs, because the first operator in Fig. 44.b would seem very desirable since it can be applied to produce

$$Y = - Z * (C - D).$$

A better difference is that the Z is in the wrong position; it should be higher in the subexpression to the left of '='. Using this difference, GPS would select and apply the first operator in Fig. 44.c producing

$$Z * (C - D) = - Y.$$

Detecting the same difference again, GPS would select and apply the third operator in Fig. 44.c to produce

$$Z = -Y/(C - D).$$

The Z in the first equation in Fig. 44.a can now be replaced yielding

$$A * Y - B * Y / (C - D) = Y + E.$$

This example illustrates that GPS must have a good set of differences to guide its search. Two types of differences were introduced that cannot

be represented in the current version of GPS: Tree structure types of differences referred to the global properties of tree structures or subtree structures, e.g., a tree structure contains too many occurrences of a particular symbol. The position type of difference which cannot be represented as a FEATURE refers to the position of a particular symbol. Such a difference denotes where a symbol should be, relative to where it is.

In GPS-2-5, the match could detect tree structure differences between two OBJECT-SCHEMAs, e.g.,

> "the number of Z's occurring in a tree structure should be decreased by one."

This facility could easily be added to GPS. However, to detect these types of differences between an OBJECT-SCHEMA and a DESCRIBED-OBJ is considerably more complex, because their modes of representation are different, and the desirability-selection process for MOVE-OPERATORS assumes that the types of differences are FEATURES. By making some of the routines in GPS more sophisticated, tree structure types of differences could be added to GPS although there probably are some unforseen difficulties.

On the other hand, position types of differences cannot be added to GPS without extensive modifications. The reason is that the match places the nodes in the tree structures of the two OBJECT-SCHEMAs into correspondence. The resulting differences which are detected are all of the form:

> At a particular position in the tree structure, certain features should have certain values.

In order to find position differences, the match must place symbols into correspondence and compare their positions in the tree structures. For example, in the two tree structures in Fig. 45, the match must place the Z's in correspondence to detect that the Z in Fig. 45.a is to the LEFT of

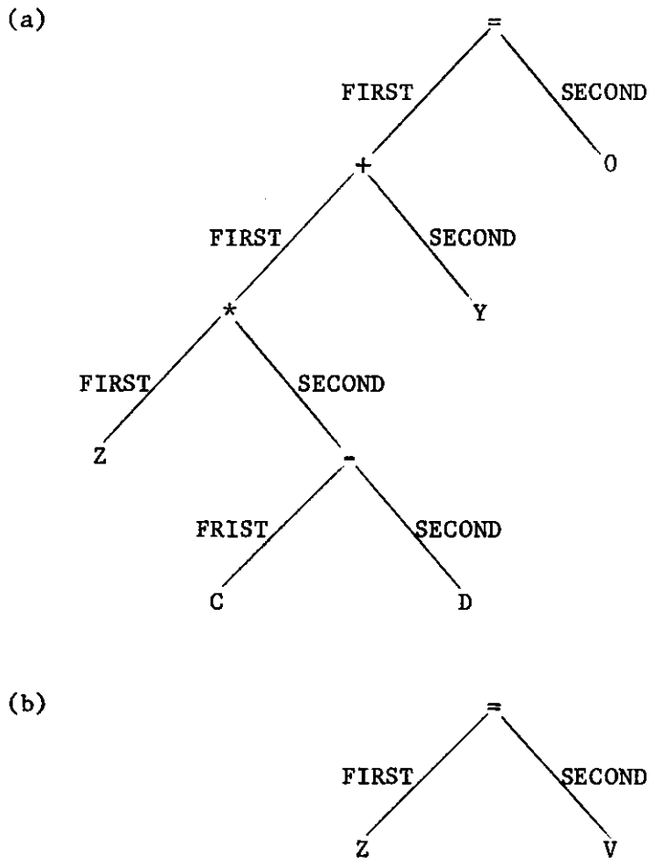


FIGURE 45. Two tree structures which are matched in solving the task in Fig. 44.

the LEFT of the Z in Fig. 45.b. Thus, both tree structure differences and position differences cannot be used because of shortcomings in the object-difference, operator-difference and desirability-selection.

G. CONCLUSION

The initial motivation for this research comes from the tasks themselves; they could not be expressed in the internal representation of GPS-2-5. But the majority of this research focuses on the processing implication of several modes of representation.

In generalizing GPS, the trend appears to be toward richer representations whose structures are simple enough for GPS to understand. The structure of test-processes is too complex for GPS to understand and thus, GPS's problem solving techniques cannot be applied to this mode of representation.

On the other hand, GPS can sufficiently understand unordered-schema and characteristic-lists-schemas to apply its problem solving techniques to them. There are difficulties. In going to characteristic-lists-schemas, the types of differences were limited to FEATURES; GPS can no longer detect the "tree structure" types of differences that GPS-2-5 detected in the logic task. In matching two unordered-schemas, GPS cannot guarantee that they cannot be made identical if it fails to find a match. However, GPS can sufficiently process these modes of representation to solve the tasks in the next chapter.

Both unordered-schemas and characteristic-list-schemas are more general representations than the schemas which were used exclusively in GPS-2-5. Either can be used to represent classes of schemas and any schema can be represented by an unordered-schema or a characteristic-list-schema. However, schemas still play an important role in the internal representation of GPS.

A DESIRED-OBJ can only be compared to an OBJECT-SCHEMA and not to another DESIRED-OBJ. Similarly a MOVE-OPERATOR can only be applied to an OBJECT-SCHEMA and can only produce an OBJECT-SCHEMA. The reason for this dichotomy is that the values of FEATUREs are given explicitly in OBJECT-SCHEMAs, whereas the values of FEATUREs in DESCRIBED-OBJ may be given by several constraints. Consequently, the process for evaluating a FEATURE can only find the values of the FEATUREs of OBJECT-SCHEMAs.

In moving to richer representations, GPS's basic knowledge had to be enlarged by increasing the variety and complexity of its basic processes. This seems to imply larger and more inefficient programs; but there is considerable evidence to the contrary. With the addition of one complex process, others may come practically free. The object-difference process for DESIRED-OBJ and the operator-difference process for MOVE-OPERATORs use much of the same apparatus. Although the operator-application process for MOVE-OPERATORs is complex, the feasibility-selection is free. All data structures are canonized by the same basic canonization process and the canonization of a new type of data structure, e.g., FEATUREs and partially specified MOVE-OPERATORs, is a trivial addition to GPS.

The processing required for richer representation may be more efficient. By using MOVE-OPERATORS without variables feasibility-selection can be applied at no additional expense to larger classes of model operators than was possible when FORM-OPERATORs were used. Similarly, the use of variables in MOVE-OPERATORs allows the desirability-selection to be applied to a much larger class of model operators than is possible when operators are represented as characteristic-lists or schemas. Although the match for unordered-schemas is considerably more expensive than the match for schemas, the space which must be searched by GPS is exponentially smaller.

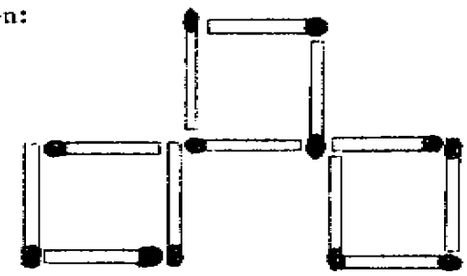
The generalization of GPS has focused on a particular group of tasks. If other tasks had been chosen, the generalization might have followed a quite different course. The tasks dealt with in this research were not chosen arbitrarily. Some categories of tasks, e.g., many optimization tasks, were deliberately avoided because we knew of no obvious heuristic search formulation for them. Other categories of tasks were avoided because of deficiencies in GPS's problem solving methods. For example, games were avoided because GPS does not have a method for considering the opponent's moves and interests in addition to its own.

The organization of GPS is particularly suitable for the addition of new mechanisms. Its problem solving techniques are segmented into a number of special purpose methods and general processes common to the application of all methods. A method for working backwards (applying the inverse operators to the desired situation and objects derived from it) could easily be added when the desired situation is an OBJECT-SCHEMA. However, it would be difficult to make this method general enough to be applicable when the desired situation is a DESCRIBED-OBJ, since operators are only applied to OBJECT-SCHEMAs. This example again illustrates that the addition of new methods increases the problem solving demands, which in turn limit the modes or representation that can be used. Hence, to keep GPS's problem solving demands constant, its methods were not augmented.

Of the tasks dealt with in this research, GPS can solve some, typified by the tasks in the next chapter. However, others were not given to GPS because of inadequacies in its representation. Examples of such tasks are board puzzles, in particular, the match-stick task in Fig. 46 and the block puzzle in Fig. 47. One difficulty, already discussed, is that the objects, which are board situations, are too large for GPS's limited memory. However,

Generic form of objects: A configuration of match-sticks on a table.

Initial situation:



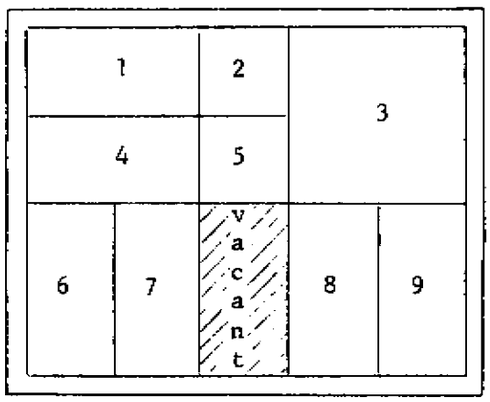
Desired situation: A configuration containing four squares that can be obtained by moving precisely three match-sticks.

Operator: Move a match-stick from one place to another.

FIGURE 46. A match- stick task.

Generic form of objects: A configuration of the nine blocks in the box.

Initial situation:



Desired situation: A configuration in which block #3 is in the lower left corner.

Operator: Move any block into an adjacent vacancy; e.g.,move block #8 into the vacancy at its left.

FIGURE 47. A block puzzle.

the main difficulty is that the types of differences (FEATUREs) of GPS are not adequate to guide GPS's search effectively, for most board puzzles.

The elementary types of differences used in GPS also prevent GPS from solving other tasks such as proving trigonometric identities, proving logic theorems, and solving simultaneous equations. Although GPS was given a logic task using the MOVE-OPERATORs in Appendix B, it could not solve the task because of its inadequate representation of differences. This task had previously been solved by GPS-2-5.

This concludes the discussion of GPS. Chapter VI demonstrates GPS's generality and ability via concrete tasks. Clearly, these tasks are a representative, and not an exhaustive, example.

CHAPTER VI: TASKS GIVEN TO GPS

This chapter discusses eleven different tasks which were given to GPS. The discussion of each task consists of four parts: a description of the task; a description of the specification of the task for GPS; a description of the way in which GPS attempts to find a solution for the task; and a discussion of the important aspects of giving the task to GPS.

All of the tasks in this chapter were run on the CDC, G21 computer. The IPL-V system on this machine has an "available space" of about 20K IPL locations. On the average, 76K IPL instructions were spent on processing a GOAL. This amounts to approximately 76 seconds per GOAL on the G21 or in more common terms (since the G21 is a rare machine), about 17 seconds per GOAL on an IBM 7090.

A. MISSIONARIES AND CANNIBALS TASK

The missionaries and cannibals task (described on page 1) has been solved by GPS-2-2[1].

GPS Formulation

The missionaries and cannibals task has already been discussed extensively[2], and only the details are given here. The task is formulated for GPS in Fig. 48. TOP-GOAL is a statement of the problem. INITIAL-OBJ, shown in tree structure form in Fig. 49, represents the situation when all of the missionaries, all of the cannibals and the BOAT are on the LEFT bank of the

---

1See pages 92 to 96 in Newell [35].

2The formulation of the M-C-OPR is described on pages 93-100; the translation of Fig. 1 is described on pages 113-5.

```
RENAME   (
   LEFT = FIRST
   RIGHT = SECOND
     )
DECLARE     (
   BOAT = ATTRIBUTE
   C = ATTRIBUTE
   B-L = FEATURE
   B-R = FEATURE
   C-L = FEATURE
   C-R = FEATURE
   DESIRED-OBJ = OBJECT-SCHEMA
   FROM-SIDE = LOC-PROG
   FROM-SIDE-TESTS = V-TESTS
   INITIAL-OBJ = OBJECT-SCHEMA
   M = ATTRIBUTE
   M-C-OPR = MOVE-OPERATOR
   M-L = FEATURE
   M-R = FEATURE
   SIDE-SET = SET
   TO-SIDE = LOC-PROG
   TO-SIDE-TESTS = V-TESTS
   X = CONSTANT
   X+Y = EXPRES
   Y = CONSTANT
   0,1,2-SET = SET
   1,2 = SET
```

Figure 48: The specification for GPS of the missionaries and cannibals task.

```
    )
TASK-STRUCTURES     (
   TOP-GOAL = ( TRANSFORM THE INITIAL-OBJ INTO THE DESIRED-OBJ ; )
   INITIAL-OBJ = ( LEFT ( M 3 C 3 BOAT YES )
                   RIGHT ( M 0 C 0 ) )
   DESIRED-OBJ = ( LEFT ( M 0 C 0 )
                    RIGHT ( M 3 C 3 BOAT YES ) )
   X+Y = ( X + Y )
   1,2 = ( 1 2 )
   0,1,2-SET = ( 0 1 2 )
   SIDE-SET = ( LEFT RIGHT )
   FROM-SIDE-TESTS = (   1. THE M OF THE FROM-SIDE IS NOT-LESS-THAN
                            THE C OF THE FROM-SIDE .
                         2. THE M OF THE FROM-SIDE EQUALS 0 . )
   TO-SIDE-TESTS = (  1. THE M OF THE TO-SIDE IS NOT-LESS-THAN
                         THE C OF THE TO-SIDE .
                      2. THE M OF THE TO-SIDE EQUALS 0 . )
   M-C-OPR = ( CREATION-OPERATOR
               $ MOVE X MISSIONARIES AND Y CANNIBALS FROM THE FROM-SIDE TO
                 THE TO-SIDE $
               VAR-DOMAIN
               1. Y IS A CONSTRAINED-MEMBER OF THE 0,1,2-SET ,
                  THE CONSTRAINT IS X+Y IS IN-THE-SET 1,2 ,
               2. X IS A CONSTRAINED-MEMBER OF THE 0,1,2-SET ,
                  THE CONSTRAINT IS X+Y IS IN-THE-SET 1,2 ,
               3. THE FROM-SIDE IS AN EXCLUSIVE-MEMBER OF THE SIDE-SET ,
               4. THE TO-SIDE IS AN EXCLUSIVE-MEMBER OF THE SIDE-SET ,
```

Figure 48: (continued)

```
            MOVES
              1. MOVE THE BOAT OF THE FROM-SIDE TO THE BOAT OF THE TO-SIDE .
              2. DECREASE BY THE AMOUNT X THE M AT THE FROM-SIDE AND ADD
                 IT TO THE M AT THE TO-SIDE .
              3. DECREASE BY THE AMOUNT Y THE C AT THE FROM-SIDE AND ADD
                 IT TO THE C AT THE TO-SIDE .
            POST-TESTS
              1. ARE ANY OF THE FROM-SIDE-TESTS TRUE .
              2. ARE ANY OF THE TO-SIDE-TESTS TRUE . )
    .
B-L = ( BOAT ON THE LEFT . )
B-R = ( BOAT ON THE RIGHT . )
C-L = ( C ON THE LEFT . )
C-R = ( C ON THE RIGHT . )
M-L = ( M ON THE LEFT . )
M-R = ( M ON THE RIGHT . )
DIFF-ORDERING = ( ( M-R M-L C-R C-L )
                  ( B-R B-L ) )
TABLE-OF-CONNECTIONS = ( ( COMMON-DIFFERENCE M-C-OPP ) )
COMPARE-OBJECTS = ( BASIC-MATCH )
BASIC-MATCH = ( COMP-FEAT-LIST ( M-L C-L B-L ) )
OBJ-ATTRIB = ( M C BOAT )
LIST-OF-VAR = ( FROM-SIDE TO-SIDE X Y )
  )     END
```

Figure 48: (continued)

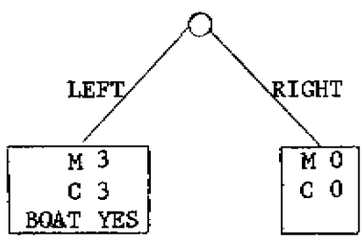


FIGURE 49. The tree structure representation of INITIAL-OBJ.

river. M is the ATTRIBUTE whose value is the number of missionaries, and C is the ATTRIBUTE whose value is the number of cannibals. The value, YES, of the ATTRIBUTE, BOAT, signifies the presence of the BOAT, while the absence of the BOAT is indicated by the absence of a value of BOAT.

DESIRED-OBJ is the OBJECT-SCHEMA that represents the situation when all of the missionaries, all of the cannibals, and the BOAT are at the RIGHT bank of the river.

The only operator in this formulation is the M-C-OPR whose application has the effect of moving X missionaries, Y cannibals and the BOAT from the FROM-SIDE to the TO-SIDE. The first two TESTs in VAR-DOMAIN require that both X and Y are either 0, 1, or 2 and that their sum is either 1 or 2. The sum of X and Y, which represents the number of people in the BOAT, cannot be less than 1 because there must be someone operating the BOAT. Since the capacity of the BOAT is two, the sum of X + Y must be no greater than two.

The third and fourth TESTs in VAR-DOMAIN designate that FROM-SIDE and TO-SIDE stand for different banks of the river. The three TRANSFORMATIONs have the effect of moving across the river the BOAT, X missionaries, and Y cannibals, respectively.

The formulation assumes that in all existing objects no missionaries are eaten and checks this constraint before producing a new object. If POST-TESTS are satisfied, no missionaries will be eaten in the resultant OBJECT-SCHEMA, because they cannot be eaten in the BOAT, due to its limited capacity. Since GPS does not understand the concept of "eating", it must be told, explicitly, that a missionary must be present before he can be eaten (the second TEST in both FROM-SIDE-TESTS and TO-SIDE-TESTS).

COMPARE-OBJECTS and BASIC-MATCH indicate that all differences are observed at the TOP-NODE, and that only the ATTRIBUTEs of the node at the

LEFT are matched. Since the formulation prevents the missionaries from being eaten, everything which is not at LEFT must be at RIGHT. If the values of corresponding ATTRIBUTEs of the RIGHT are also matched, the differences detected would cause the same operators to be selected.

DIFF-ORDERING designates that types of differences that pertain to the BOAT are easier than those that pertain to the missionaries or cannibals. TABLE-OF-CONNECTIONS signifies that the M-C-OPR is relevant to reducing all differences and does not contain any information about the desirability of operators to reducing particular types of differences. OBJ-ATTRIB lists the ATTRIBUTEs of this task and LIST-OF-VAR lists the variables that appear in Fig. 49.

NEW-OBJ is a criterion for selecting OBJECT-SCHEMAs from a SET of OBJECT-SCHEMAs and does not appear in Fig. 48, because it was given to GPS as an IPL-V structure. This criterion is used in conjunction with the TRANSFORM-SET-METHOD (see Fig. 25). The first submethod is a SELECT GOAL-SCHEMA whose selection criterion is NEW-OBJ in this task as well as in several other tasks (water jug task, father and sons task). However, a different criterion may be used. For example, the predicate calculus task uses SMALLEST (see page 220) as the criterion.

Any OBJECT-SCHEMA that has never appeared in the statement of a TRANSFORM GOAL fulfills the NEW-OBJ criterion. GPS attempts to generate new GOALs (EXPANDED-TRANSFORM-METHOD) whenever it runs into trouble; e.g., enters a loop by generating a GOAL equivalent to an old GOAL. Transforming an OBJECT-SCHEMA that fulfills NEW-OBJ into the desired object is a new GOAL and attempting it might yield new results.

The TRANSFORM-METHOD has been slightly generalized for this task (and this generalized version is also used in the Bridges of Königsberg). Whenever

the MATCH-DIFF-METHOD detects more than one difference with the same difficulty according to DIFF-ORDERING, GPS has the capability of generating more than one REDUCE GOAL. The rationale is that since GPS has no reason for selecting any difference in particular, it considers them all, if necessary. The TRANSFORM-METHOD (shown in Fig. 17) terminates on the FAILURE of the REDUCE GOAL-SCHEMA method (fifth submethod). In the generalized version of this method, FAILURE of the REDUCE GOAL-SCHEMA method causes GPS to attempt to construct another REDUCE GOAL.

Behavior of GPS

Fig. 50 shows the behavior of GPS in solving the task in Fig. 48. In attempting TOP-GOAL GPS detects that there are too many missionaries and cannibals at the LEFT and that the BOAT should not be at the LEFT. GOAL 2 is created in an attempt to reduce the number of cannibals at the LEFT and the M-C-OPR with Y equal to 2 and FROM-SIDE equal to LEFT is considered relevant to reducing this difference. GOAL 3 results in OBJECT 5, and GPS attempts to transform the new object into the DESIRED-OBJ (GOAL 4).

Since there are still too many cannibals at the LEFT (GOAL 5), GPS attempts to move the remaining cannibal to the RIGHT (GOAL 6). In an attempt to bring the BOAT back to the LEFT (GOAL 7), GPS applies the M-C-OPR with the TO-SIDE equal to LEFT (GOAL 8) and OBJECT 6 is produced. GPS moves one cannibal across the river (GOAL 9); it does not realize that bringing the BOAT back to the LEFT also brought a cannibal with it. Since transforming the result of GOAL 9, which is an old object, into the DESIRED-OBJ is an old GOAL, GPS does not attempt it but looks for something else to do.

GOAL 11 is created in an attempt to transform all of the OBJECT-SCHEMAs, which are derived from the INITIAL-OBJ, into the DESIRED-OBJ. (OBJECT 4,

```
1 TOP-GOAL TRANSFORM INITIAL-OBJ INTO DESIRED-OBJ      (SUBGOAL OF NONE)

  2 GOAL 2 REDUCE C-L ON INITIAL-OBJ      (SUBGOAL OF TOP-GOAL)

    3 GOAL 3 APPLY M-C-OPR WITH  Y = 2, FROM-SIDE = LEFT, TO INITIAL-OBJ      (SUBGOAL OF 2)
                  SET:  X = 0, TO-SIDE = RIGHT
                  OBJECT 5: (LEFT(M 3 C 1) RIGHT(M 0 C 2 BOAT YES))
  2  GOAL  4 TRANSFORM 5 INTO DESIRED-OBJ      (SUBGOAL OF TOP-GOAL)

    3 GOAL 5 REDUCE C-L ON 5      (SUBGOAL OF 4)

      4 GOAL 6 APPLY M-C-OPR WITH  Y = 1, FROM-SIDE = LEFT, TO 5      (SUBGOAL OF 5)
                  SET:  X = 0, TO-SIDE = RIGHT

        5 GOAL 7 REDUCE B-L ON 5      (SUBGOAL OF 6)

          6 GOAL 8 APPLY M-C-OPR WITH  TO-SIDE = LEFT, TO 5      (SUBGOAL OF 7)
                  SET:  Y = 1, X = 0, FROM-SIDE = RIGHT
                  OBJECT 6: (LEFT(M 3 C 2 BOAT YES) RIGHT(M 0 C 1))

        5 GOAL 9 APPLY M-C-OPR WITH  Y = 1, FROM-SIDE = LEFT, TO 6      (SUBGOAL OF 6)
                  SET:  X = 0, TO-SIDE = RIGHT
                  OBJECT 5: (LEFT(M 3 C 1) RIGHT(M 0 C 2 BOAT YES))
  2  GOAL  11 TRANSFORM 4 INTO DESIRED-OBJ      (SUBGOAL OF TOP-GOAL)

    3 GOAL 12 SELECT FROM 4 A/C NEW-OBJ OF DESIRED-OBJ      (SUBGOAL OF 11)
                  A SELECTED

    3  GOAL  13 TRANSFORM A INTO DESIRED-OBJ      (SUBGOAL OF 11)

      4 GOAL 14 REDUCE C-L ON A      (SUBGOAL OF 13)

        5 GOAL 15 APPLY M-C-OPR WITH  Y = 2, FROM-SIDE = LEFT, TO A      (SUBGOAL OF 14)
                  SET:  X = 0, TO-SIDE = RIGHT
                  OBJECT 7: (LEFT(M 3 C 0) RIGHT(M 0 C 3 BOAT YES))

      4  GOAL  16 TRANSFORM 7 INTO DESIRED-OBJ      (SUBGOAL OF 13)

        5 GOAL 17 REDUCE M-L ON 7      (SUBGOAL OF 16)
```

Figure 50: The performance of GPS on the missionaries and cannibals task.

```
5 GOAL 19 APPLY M-C-OPR WITH  X = 2, FROM-SIDE = LEFT, TO 7      (SUBGOAL OF 17)
          SET:  Y = 0, TO-SIDE = RIGHT

  7 GOAL 19 REDUCE B-L ON 7      (SUBGOAL OF 18)

    8 GOAL 20 APPLY M-C-OPR WITH  TO-SIDE = LEFT, TO 7      (SUBGOAL OF 19)
          SET:  Y = 1, X = 0, FROM-SIDE = RIGHT
          OBJECT 8: (LEFT(M 3 C 1 BOAT YES) RIGHT(M 0 C 2))

    7 GOAL 21 APPLY M-C-OPR WITH  X = 2, FROM-SIDE = LEFT, TO 8      (SUBGOAL OF 18)
          SET:  Y = 0, TO-SIDE = RIGHT
          OBJECT 9: (LEFT(M 1 C 1) RIGHT(M 2 C 2 BOAT YES))

5 GOAL 22 TRANSFORM 9 INTO DESIRED-OBJ      (SUBGOAL OF 16)

  5 GOAL 23 REDUCE C-L ON 9      (SUBGOAL OF 22)

    7 GOAL 24 APPLY M-C-OPR WITH  Y = 1, FROM-SIDE = LEFT, TO 9      (SUBGOAL OF 23)
          SET:  X = 0, TO-SIDE = RIGHT

      8 GOAL 25 REDUCE B-L ON 9      (SUBGOAL OF 24)

        9 GOAL 26 APPLY M-C-OPR WITH  TO-SIDE = LEFT, TO 9      (SUBGOAL OF 25)
          SET:  Y = 1, X = 1, FROM-SIDE = RIGHT
          OBJECT 10: (LEFT(M 2 C 2 BOAT YES) RIGHT(M 1 C 1))

      8 GOAL 27 APPLY M-C-OPR WITH  Y = 1, FROM-SIDE = LEFT, TO 10      (SUBGOAL OF 24)
          SET:  X = 1, TO-SIDE = RIGHT
          OBJECT 9: (LEFT(M 1 C 1) RIGHT(M 2 C 2 BOAT YES))

3 GOAL 12 SELECT FROM 4 A/C NEW-OBJ OF DESIRED-OBJ      (SUBGOAL OF 11)
          10 SELECTED

3 GOAL 29 TRANSFORM 8 INTO DESIRED-OBJ      (SUBGOAL OF 11)

  4 GOAL 30 REDUCE C-L ON 8      (SUBGOAL OF 29)

    5 GOAL 31 APPLY M-C-OPR WITH  Y = 1, FROM-SIDE = LEFT, TO 8      (SUBGOAL OF 30)
          SET:  X = 0, TO-SIDE = RIGHT
          OBJECT 7: (LEFT(M 3 C 0) RIGHT(M 0 C 3 BOAT YES))

3 GOAL 12 SELECT FROM 4 A/C NEW-OBJ OF DESIRED-OBJ      (SUBGOAL OF 11)
          10 SELECTED

3 GOAL 33 TRANSFORM 10 INTO DESIRED-OBJ      (SUBGOAL OF 11)
```

Figure 50: (continued)

```
4 GOAL 34 REDUCE C-L ON 10      (SUBGOAL OF 33)

  5 GOAL 35 APPLY M-C-OPR WITH  Y = 2, FROM-SIDE = LEFT, TO 10      (SUBGOAL OF 34)
            SET:  X = 2, TO-SIDE = RIGHT

4 GOAL 36 REDUCE M-L ON 10      (SUBGOAL OF 33)

  5 GOAL 37 APPLY M-C-OPR WITH  X = 2, FROM-SIDE = LEFT, TO 10      (SUBGOAL OF 36)
            SET:  Y = 0, TO-SIDE = RIGHT
            OBJECT 11: (LEFT(M 0 C 2) RIGHT(M 3 C 1 BOAT YES))

4 GOAL 38 TRANSFORM 11 INTO DESIRED-OBJ      (SUBGOAL OF 33)

  5 GOAL 39 REDUCE C-L ON 11      (SUBGOAL OF 38)

    6 GOAL 40 APPLY M-C-OPR WITH  Y = 2, FROM-SIDE = LEFT, TO 11      (SUBGOAL OF 39)
              SET:  X = 0, TO-SIDE = RIGHT

      7 GOAL 41 REDUCE B-L ON 11      (SUBGOAL OF 40)

        8 GOAL 42 APPLY M-C-OPR WITH  TO-SIDE = LEFT, TO 11      (SUBGOAL OF 41)
                  SET:  Y = 1, X = 0, FROM-SIDE = RIGHT
                  OBJECT 12: (LEFT(M 0 C 3 BOAT YES) RIGHT(M 3 C 0))

      7 GOAL 43 APPLY M-C-OPR WITH  Y = 2, FROM-SIDE = LEFT, TO 12      (SUBGOAL OF 40)
                SET:  X = 0, TO-SIDE = RIGHT
                OBJECT 13: (LEFT(M 0 C 1) RIGHT(M 3 C 2 BOAT YES))

4 GOAL 44 TRANSFORM 13 INTO DESIRED-OBJ      (SUBGOAL OF 38)

  5 GOAL 45 REDUCE C-L ON 13      (SUBGOAL OF 44)

    6 GOAL 46 APPLY M-C-OPR WITH  Y = 1, FROM-SIDE = LEFT, TO 13      (SUBGOAL OF 45)
              SET:  X = 0, TO-SIDE = RIGHT

      8 GOAL 47 REDUCE B-L ON 13      (SUBGOAL OF 46)

        9 GOAL 48 APPLY M-C-OPR WITH  TO-SIDE = LEFT, TO 13      (SUBGOAL OF 47)
                  SET:  Y = 1, X = 0, FROM-SIDE = RIGHT
                  OBJECT 14: (LEFT(M 0 C 2 BOAT YES) RIGHT(M 3 C 1))

      8 GOAL 49 APPLY M-C-OPR WITH  Y = 1, FROM-SIDE = LEFT, TO 14      (SUBGOAL OF 46)
```

Figure 50: (continued)

```
                    SET:  X = 0, TO-SIDE = RIGHT
                    OBJECT 13: (LEFT(M 0 C 1) RIGHT(M 3 C 2 BOAT YES))

3 GOAL 12 SELECT FROM 4 A/C NEW-OBJ OF DESIRED-OBJ      (SUBGOAL OF 11)
                    14 SELECTED

3  GOAL  51 TRANSFORM 12 INTO DESIRED-OBJ      (SUBGOAL OF 11)

  4 GOAL 52 REDUCE C-L ON 12      (SUBGOAL OF 51)

3 GOAL 12 SELECT FROM 4 A/C NEW-OBJ OF DESIRED-OBJ      (SUBGOAL OF 11)
                    14 SELECTED

3  GOAL  54 TRANSFORM 14 INTO DESIRED-OBJ      (SUBGOAL OF 11)

  4 GOAL 55 REDUCE C-L ON 14      (SUBGOAL OF 54)

    5 GOAL 56 APPLY M-C-OPP WITH  Y = 2, FROM-SIDE = LEFT, TO 14      (SUBGOAL OF 55)
                    SET:  X = 0, TO-SIDE = RIGHT
                    OBJECT 15: (LEFT(M 0 C 0) RIGHT(M 3 C 3 BOAT YES))

  4  GOAL  57 TRANSFORM 15 INTO DESIRED-OBJ      (SUBGOAL OF 54)
SUCCESS
```

Figure 50: (continued)

which is the SET of all OBJECT-SCHEMAs derived from the INITIAL-OBJ, is generated internally by GPS). GOAL 13 is created because OBJECT 6 has never appeared in a TRANSFORM GOAL. Since there are too many cannibals at the LEFT in OBJECT 6 (GOAL 14), two are moved across the river (GOAL 15, OBJECT 7).

Everything goes smoothly until GOAL 27, which results in an old object, at which point GPS generates a GOAL identical to GOAL 22. GPS does not reattempt GOAL 22 but generates a new GOAL (GOAL 29) by selecting a NEW-OBJ (OBJECT 8). Attempting GOAL 29 quickly leads to the old object, OBJECT 7 (GOAL 31) and the old GOAL, GOAL 16.

GOAL 33 is generated by selecting another NEW-OBJ and GPS does not run into trouble until GOAL 49 which results in an old object. Again a new GOAL is generated by selecting a NEW-OBJ. GOAL 52 is abandoned because attempting it creates a GOAL identical to GOAL 43. The generation of GOAL 54 quickly leads to success.

Discussion

It is instructive to compare the specification of the missionaries and cannibals task in Fig. 48 with its specification for GPS-2-2[3]. The latter contains information about the nature of operators which the current GPS discovers for itself. GPS-2-2 was given ten operators: Move one missionary from left to right; move two missionaries from left to right; move one missionary and one cannibal from left to right, etc. The desirability of these operators for reducing the various types of differences was given to GPS-2-2 exogenously in the TABLE-OF-CONNECTIONS. In Fig. 48 there is only a single operator. In applying this operator, GPS specifies the variables so that the operator performs a desirable function.

[3]See pages 92 to 96 in Newell [35].

GPS-2-2 was given a desirability filter for operators. This filter prevented GPS-2-2 from attempting to move more missionaries and cannibals across the river than there were on the side from which they were being moved. Such a separate filter is unnecessary in GPS because GPS never considers applying such an operator. Each operator in the GPS-2-2 formulation consisted of an IPL routine with its parameters (described on pages 30-3). The operator filter was also encoded in IPL. Not only is it tedious to construct IPL routines but the construction of these routines required some knowledge of the internal structure of GPS. The M-C-OPR in Fig. 48 contains no information about the internal structure of GPS.

B. INTEGRATION

This task is analogous to that faced by an engineer who wants to integrate an expression symbolically. If the expression is at all complex, the engineer will probably use an integration table; otherwise, he will use an elementary integral form which he has memorized. Although all of these forms can be derived either by limiting procedures or from previously derived forms, it is impractical for the engineer or GPS to do so.

GPS Formulation

Only the details of integration are described here because this task has already been discussed extensively. Fig. 51 is the GPS formulation of the task of integrating EXPRESSION-1, shown in tree structure form in Fig. 52, which represents the integral,

$$\int te^{t^2}\,dt.$$

TWO is used instead of '2' because TWO is a symbolic CONSTANT and not a number (IPL data term). By convention a value of SYMBOL cannot be a number.

```
RENAME   (
   LEFT = FIRST
   RIGHT = SECOND
     )
DECLARE   (
   COS = UNARY-CONNECTIVE
   D = UNARY-CONNECTIVE
   DESIRED-OBJ = DESCRIBED-OBJ
   EXP = N-ARY-CONNECTIVE
   INTEGRAL = UNARY-CONNECTIVE
   LOG = UNARY-CONNECTIVE
   SIN = UNARY-CONNECTIVE
   SYMBOL = ATTRIBUTE
   SYM-DIFF = FEATURE
   - = UNARY-CONNECTIVE
   * = N-ARY-CONNECTIVE
   + = N-ARY-CONNECTIVE
     )
LIST     (
   EXPRESSION-1 = ( THE INTEGRAL OF ( T * ( E EXP ( T EXP TWO ) ) * D T ) )
   INTEGRATE = ( 1. ( THE INTEGRAL OF ( ( U EXP N ) * D U ) YIELDS
                    ( ( U EXP ( N + ONE ) ) * ( ( N + ONE ) EXP - ONE ) ) ) ,
                2. ( THE INTEGRAL OF ( ( U EXP - ONE ) * D U ) YIELDS
                   LOG U ) ,
                3. ( THE INTEGRAL OF ( SIN U * D U ) YIELDS - COS U ) ,
                4. ( THE INTEGRAL OF ( COS U * D U ) YIELDS SIN U ) ,
                5. ( THE INTEGRAL OF ( U * D U ) YIELDS ( ( U EXP TWO ) *
```

Figure 51: The specification for GPS of the task of integrating $\int te^{t^2} dt$.

```
                    ( TWO EXP - ONE ) ) ) ,
                 6. ( THE INTEGRAL OF ( ( E EXP U ) * D U ) YIELDS
                    ( E EXP U ) ) ,
                 7. ( THE INTEGRAL OF ( ( F + G ) * D U ) YIELDS ( THE INTEGRAL
                    OF ( F * D U ) + THE INTEGRAL OF ( G * D U ) ) ) )
   DIFFERENTIATE = ( 1. ( ( SIN U * D U ) YIELDS - U COS U ) ,
                    2. ( ( COS U * D U ) YIELDS D SIN U ) ,
                    3. ( ( U * D U ) YIELDS ( ( TWO EXP - ONE ) *
                       U ( U EXP TWO ) ) ) ,
                    4. ( ( ( U EXP - ONE ) * D U ) YIELDS D LOG U ) )
   EXPRESSION-2 = ( THE INTEGRAL OF ( ( ( ( SIN ( C * T ) EXP TWO ) *
                  COS ( C * T ) ) * ( T EXP - ONE ) ) * D T ) )
     )
TASK-STRUCTURES   (
   TOP-GOAL = ( TRANSFORM EXPRESSION-1 INTO THE DESIRED-OBJ . )
   DESIRED-OBJ = ( SUBEXPRESSION-TESTS
                   THE SYMBOL DOES NOT-EQUAL INTEGRAL . )
   SYM-DIFF = ( SYMBOL )
   TABLE-OF-CONNECTIONS = ( 1. ( SET-SIZE DIFFERENTIATE )
                            2. ( SYM-DIFF INTEGRATE ) )
   DIFF-ORDERING = ( 1. SYM-DIFF
                     2. SET-SIZE )
   LIST-OF-OPS = ( INTEGRATE DIFFERENTIATE )
   LIST-OF-VARS = ( F G U N )
   )
   END
```

Figure 51: (continued)

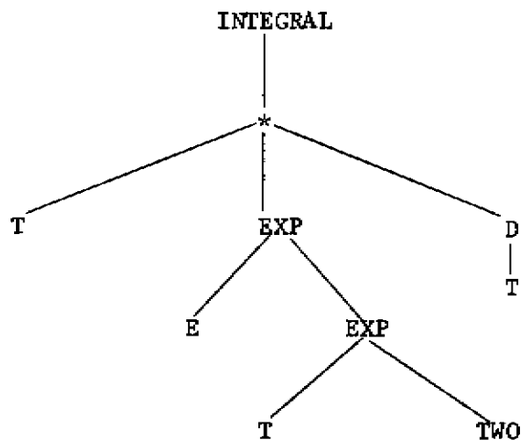


FIGURE 52. The tree structure representation of EXPRESSION-1. The symbols at the nodes are values of the ATTRIBUTE, SYMBOL of the node.

The only numbers used are integers. Fractions are represented by the use of EXP. For instance, 2/3 is represented as TWO * (THREE EXP - ONE). EXPRESSION-1 and the operators of this task are translated as a linear list of symbols (LIST mode) after translation. The DESIRED-OBJ represents an expression which does not contain an '$\int$', i.e., at every node the value of the ATTRIBUTE, SYMBOL, is not INTEGRAL. TOP-GOAL is the statement that the problem is to remove all occurrences of INTEGRAL from EXPRESSION-1.

In this task there are only two operators--INTEGRATE and DIFFERENTIATE--both of which are a SET of FORM-OPERATORs separated by commas. INTEGRATE represents an integral table, since each of the FORM-OPERATORs which it contains is a standard integral form. For example, the first FORM-OPERATOR in INTEGRATE represents the integral form,

$$\int u^n \, du = \frac{u^{n+1}}{n+1}$$

Similarly, DIFFERENTIATE represents a table of standard derivatives.

In addition to these two explicit operators, there are several operators given to GPS as IPL structures which will be discussed in detail later. These operators, listed below, can be applied implicitly; i.e., without creating special GOALs to apply them:

a. commutativity and associativity of addition
b. commutativity and associativity of multiplication
c. $\int c\ f(u)\ du = c \int f(u)\ du$
d. d (c f(u)) = c d (f(u))
e. d (c + f(u)) = d (f(u))
f. u = v implies that wu = wv
g. numeric simplification: u * 0 = 0, u * 1 = u, u + 0 = 0, u EXP 0 = 1, and integer arithmetic.

There are two types of differences in this task--SYM-DIFF and SET-SIZE. SYM-DIFF indicates that the value of SYMBOL of the node where the difference

was detected is incorrect. Since this type of difference is detected when an OBJECT-SCHEMA containing the SYMBOL, INTEGRAL, is compared to DESIRED-OBJ, the TABLE-OF-CONNECTIONS indicates that INTEGRATE should be used to reduce SYM-DIFF. SYM-DIFF could arise in other situations in which INTEGRATE would not be a good operator to apply; however, this formulation is sufficient for the integration tasks considered.

The other type of difference, SET-SIZE, is not a FEATURE and its role will be discussed later. For the tasks considered, DIFFERENTIATE should be applied when SET-SIZE is detected. DIFF-ORDERING designates that SYM-DIFF is the more difficult type of difference, because the main GOAL of integration is to remove all occurrences of INTEGRAL in an OBJECT-SCHEMA.

LIST-OF-OPR is a list of those operators which must be converted into their internal representation after translation. This conversion process assigns print names to each FORM-OPERATOR in INTEGRATE and DIFFERENTIATE. Fig. 53 gives names which were assigned. For example, the top-most integral form in Fig. 53 was assigned the name '1'.

SIMILARITY, which was given to GPS as an IPL structure, is a selection criterion used in SELECT type of GOALs and is a definition of the similarity of two OBJECT-SCHEMAs. It is a list of properties, ranked in order of their importance, that an OBJECT-SCHEMA might contain. All of the properties are properties of a SET of factors which contain a derivative, e.g., D T. In the order of their importance the properties are:

a. contains a factor whose SYMBOL is +;
b. contains a factor whose SYMBOL is LOG;
c. contains a factor whose SYMBOL is EXP and whose first argument is the same as its correspondent in the criterion OBJECT-SCHEMA, and whose second argument is a function of the variable of integration.

```
15 DERIVATION LIST OF (EXPRESSION-1 )                                                    335426

1 (INTEGRAL((U EXP N) * D U) YIELDS ((U EXP (N + ONE)) * ((N + ONE) EXP -ONE)))          349838

2 (INTEGRAL((J EXP -ONE) * D U) YIELDS LOG J)                                            356730

3 (INTEGRAL(SIN J * D U) YIELDS -COS J)                                                  362479

4 (INTEGRAL(COS J * D U) YIELDS SIN U)                                                   368046

5 (INTEGRAL(U * D U) YIELDS ((U EXP TWO) * (TWO EXP -ONE)))                              377697

6 (INTEGRAL((E EXP U) * D U) YIELDS (E EXP J))                                           385537

7 (INTEGRAL((F + G) * D U) YIELDS (INTEGRAL(F * D U) + INTEGRAL(G * D U)))               397877

9 ((SIN J * D U) YIELDS -D COS U)                                                        403310

10 ((COS U * D J) YIELDS D SIN U)                                                        408546

11 ((J * D U) YIELDS ((TWO EXP -ONE) * D(J EXP TWO)))                                   417861

12 (((U EXP -ONE) * D U) YIELDS D LOG U)                                                 424422
```

Figure 53: The print-name assignment of the FORM-OPERATORs in DIFFERENTIATE and INTEGRATE.

d. contains a factor whose SYMBOL is EXP and whose second argument is the same as its correspondent in the criterion OBJECT-SCHEMA, and whose second argument is a function of the variable of integration.
e. contains a factor whose SYMBOL is SIN;
f. contains a factor whose SYMBOL is COS;
g. contains none of the preceding properties.

Although this definition of similarity might seem rather strange[4], its usefulness is clarified by Fig. 54 and Fig. 55.

Behavior of GPS

Fig. 54 shows the behavior exhibited by GPS in integrating EXPRESSION-1 as formulated in Fig. 51. In attempting TOP-GOAL GPS notices that the SYMBOL at the TOP-NODE of EXPRESSION-1 is INTEGRAL and it creates GOAL 2 to change the value of SYMBOL. According to TABLE-OF-CONNECTIONS, INTEGRATE is a desirable operator and GOAL 3 is created.

Since INTEGRATE is a SET of FORM-OPERATORs, GOAL 4 is created to select one whose input form is similar to EXPRESSION-1. GPS selected OPERATOR 6 (see Fig. 53) because it, like EXPRESSION-1, contains an E raised to a power which is a function of the variable of integration. In applying OPERATOR 6 (GOAL 5) GPS substitutes T EXP TWO for U, which changes OPERATOR 6 to

$$\int e^{t^2}\, dt^2 = e^{t^2},$$

and notices that it cannot be applied directly to EXPRESSION-1 because there are too many factors in the SET of factors which is the argument of INTEGRAL in EXPRESSION 1. Since T and D T are unmatched factors (have no correspondents in OPERATOR 6), GPS creates OBJECT 17 and attempts to reduce the number of

[4]Analogous to SIMILARITY is the classification of integral forms in an integral table, e.g., logarithmic forms, exponential forms, etc.

```
1 TOP-GOAL TRANSFORM EXPRESSION-1 INTO DESIRED-OBJ      (SUBGOAL OF NONE)

  2 GOAL 2 REDUCE SYM-DIFF ON EXPRESSION-1      (SUBGOAL OF TOP-GOAL)

    3 GOAL 3 APPLY INTEGRATE TO EXPRESSION-1      (SUBGOAL OF 2)

      4 GOAL 4 SELECT FROM INTEGRATE A/C SIMILARITY OF EXPRESSION-1      (SUBGOAL OF 3)
                4 SELECTED

      4 GOAL 5 APPLY 6 TO EXPRESSION-1      (SUBGOAL OF 3)

        5 GOAL 6 REDUCE SET-SIZE ON  LEFT EXPRESSION-1      (SUBGOAL OF 5)
                OBJECT 17: (T * D T)

          6 GOAL 7 APPLY DIFFERENTIATE TO 17      (SUBGOAL OF 6)

            7 GOAL 8 SELECT FROM DIFFERENTIATE A/C SIMILARITY OF 17      (SUBGOAL OF 7)
                11 SELECTED

            7 GOAL 9 APPLY 11 TO 17      (SUBGOAL OF 7)
                OBJECT 19: ((TWO EXP -ONE) * D(T EXP TWO))
                OBJECT 20: INTEGRAL((E EXP (T EXP TWO)) * D(T EXP TWO))

        5 GOAL 10 APPLY 6 TO 20      (SUBGOAL OF 5)
                OBJECT 21: ((E EXP (T EXP TWO)) * (TWO EXP -ONE))

  2 GOAL 11 TRANSFORM 21 INTO DESIRED-OBJ      (SUBGOAL OF TOP-GOAL)

SUCCESS
```

Figure 54: The performance of GPS on the task of integrating $\int te^{t^2} dt$.

```
1 TOP-GOAL TRANSFORM EXPRESSION-2 INTO DESIRED-OBJ      (SUBGOAL OF NONE)

  2 GOAL 3 REDUCE SYM-DIFF ON EXPRESSION-2      (SUBGOAL OF TOP-GOAL)

    3 GOAL 4 APPLY INTEGRATE TO EXPRESSION-2      (SUBGOAL OF 3)

      4 GOAL 5 SELECT FROM INTEGRATE A/C SIMILARITY OF EXPRESSION-2      (SUBGOAL OF 4)
                7 SELECTED

      4 GOAL 6 APPLY 7 TO EXPRESSION-2      (SUBGOAL OF 4)
                OBJECT 20: (INTEGRAL((SIN(C * T) EXP TWO) * D T * COS(C * T)) + INTEGRAL((T EXP
                -ONE) * D T))

  2 GOAL 7 TRANSFORM 20 INTO DESIRED-OBJ      (SUBGOAL OF TOP-GOAL)

    3 GOAL 8 REDUCE SYM-DIFF ON  LEFT 20      (SUBGOAL OF 7)

      4 GOAL 9 APPLY INTEGRATE TO  LEFT 20      (SUBGOAL OF 8)

        5 GOAL 10 SELECT FROM INTEGRATE A/C SIMILARITY OF  LEFT 20      (SUBGOAL OF 9)
                7 SELECTED

        5 GOAL 11 APPLY 1 TO  LEFT 20      (SUBGOAL OF 9)

          6 GOAL 12 REDUCE SET-SIZE ON  LEFT  LEFT 20      (SUBGOAL OF 11)
                OBJECT 21: (COS(C * T) * D T)

            7 GOAL 13 APPLY DIFFERENTIATE TO 21      (SUBGOAL OF 12)

              8 GOAL 14 SELECT FROM DIFFERENTIATE A/C SIMILARITY OF 21      (SUBGOAL OF 13)
                10 SELECTED

              8 GOAL 15 APPLY 10 TO 21      (SUBGOAL OF 13)
                OBJECT 23: (D SIN(C * T) * (C EXP -ONE))
                OBJECT 24: (INTEGRAL((SIN(C * T) EXP TWO) * D SIN(C * T) * (C EXP -ONE)) + INTEGRAL
                ((T EXP -ONE) * D T))
```

Figure 55: The performance of GPS on the task of integrating $\int(\sin^2(ct)\cos(ct) + t^{-1})\,dt$.

```
      6 GOAL 16 APPLY 1 TO  LEFT 24      (SUBGOAL OF 11)
                OBJECT 25: (INTEGRAL((T EXP -ONE) * D T) + ((SIN(C * T) EXP THREE) * (THREE EXP
                           -ONE) * (C EXP -ONE)))

   3  GOAL 17 TRANSFORM 25 INTO DESIRED-OBJ      (SUBGOAL OF 7)

     4 GOAL 18 REDUCE SYM-DIFF ON  LEFT 25      (SUBGOAL OF 17)

       5 GOAL 19 APPLY INTEGRATE TO  LEFT 25      (SUBGOAL OF 18)

         6 GOAL 20 SELECT FROM INTEGRATE A/C SIMILARITY OF  LEFT 25      (SUBGOAL OF 19)
                   2 SELECTED

         6 GOAL 21 APPLY 2 TO  LEFT 25      (SUBGOAL OF 19)
                   OBJECT 26: (LOG T + ((SIN(C * T) EXP THREE) * (THREE EXP -ONE) * (C EXP -ONE))
                              )

     4  GOAL 22 TRANSFORM 26 INTO DESIRED-OBJ      (SUBGOAL OF 17)

SUCCESS
```

Figure 55: (continued)

factors in OBJECT 17.

Since DIFFERENTIATE has the capability of reducing the number of factors in a SET of factors, GOAL 7 is created. GPS does not attempt to reduce the number of factors in EXPRESSION-1, because the exponential factor might be eliminated. The result of GOAL 8 is OPERATOR 11 (see Fig. 3), because the input forms of each of the other FORM-OPERATORs in DIFFERENTIATE are more complex than OBJECT 17. That is, they contain a factor which is a function of the variable of integration; e.g., cos u, in OPERATOR 10 or u^{-1} in OPERATOR 12. Operator 11 is successfully applied to OBJECT 17 (GOAL 11), which results in OBJECT 19. OBJECT 19 is substituted for 'T * DT' in EXPRESSION-1 (OBJECT 20) because OBJECT 19 is derived from OBJECT 17.

In reattempting GOAL 5, OBJECT 21 is produced by applying 6 to OBJECT 20 (GOAL 10). GOAL 11 and, thus, TOP-GOAL are successful because there is no INTEGRAL in OBJECT 21.

Fig. 55 is the behavior of GPS when given the task of integrating EXPRESSION-2:

$$\int (\sin^2(ct)\cos(ct) + t^{-1})\,dt$$

Fig. 51 is the formulation of the task for GPS except that EXPRESSION-1 is replaced by EXPRESSION-2 in TOP-GOAL. The assignment of names in Fig. 53 is the same for this task.

Discussion

It might seem that most of the problem solving in this task is done by implicit application of the operators, listed on page 182, rather than by the explicit application of DIFFERENTIATE and INTEGRATE. Indeed, to integrate

$$\int 4*(\cos(2*t) * dt)$$

requires the implicit application of at least five operators in addition to the explicit application

$$\int \cos u * du = \sin u$$

in order to produce the expression

$$2*\sin(2*t).$$

If all of the operators were used explicitly, a tree would have to be searched to a depth of six in order to find the solution to this simple problem.

GPS's formulation of integration is quite similar to the one used by SAINT (Slagle [59]), a program which is rather proficient at symbolic integration. SAINT can apply "algorithmlike transformations" whenever necessary without creating special goals for applying them. These "algorithmlike transformations" include all of the operators used implicitly by GPS plus many others such as OPERATOR 9 in Fig. 53. Hence, both SAINT and GPS would integrate

$$\int 4* (\cos (2*t) * dt)$$

in a single step by recognizing that it is a substitution instance of

$$\int \cos u * du = \sin u.$$

In addition to the "algorithmlike transformations", SAINT can apply "heuristic transformations" which it uses in much the same way that GPS uses the operators, DIFFERENTIATE and INTEGRATE. Consider the example of integrating EXPRESSION-1 in Fig. 51. SAINT notices that integral is not of "standard form" and selects a "heuristic transformation" relevant to reducing this difficulty. The particular "heuristic transformation" selected is: Let u equal a nonconstant, nonlinear subexpression of the integrand. Although Slagle thought that, using this "heuristic transformation", SAINT would substitute $u = t^2$, SAINT actually makes the substitution $u = e^{t^2}$. Either of

these substitutions reduces the integral to "standard form" and the task is solved.

The substitution, $u = t^2$, performs the same function performed by applying OPERATOR 11 in Fig. 53 since both reduce EXPRESSION-1 to the standard integral form, $\int e^u \, du$. However, the "heuristic transformations" of SAINT are more general than the operators of GPS because each "heuristic-transformation" corresponds to many operators. This is a non-trivial difference in the two formulations.

The operators that GPS uses implicitly have two outstanding qualities: They are relatively simple compared with the operators in Fig. 53 and it is obvious when they should be applied. For example, in GOAL 15 in Fig. 55, it is obvious that

$$d(c\ t) = cdt$$

must be used after C * T has been substituted for U.

The associativity and commutativity of addition and multiplication is implicit in the representation. (See section D of Chapter V.) Multiplication and addition are represented as a function of an arbitrary number of arguments, and are represented as an unordered set. In matching two unordered sets, GPS pairs the elements so that the difference between the two sets will be small. Even though in Fig. 54 and in Fig. 55 the elements in unordered sets always appear in the correct order, GPS pairs the elements without any consideration to their order[5].

[5]In listing the elements of a set, they must be listed in some order. But the set can be considered unordered if no significance is attached to the order in which the elements are listed.

The commutative match, used in this task, pairs elements according to their "importance" (see pages 145-50). The most important element is the one which best fulfills the criterion, SIMILARITY (see pages 183-5). However, a SELECT GOAL is not constructed to select the most important element; the commutative match uses this criterion in a special way.

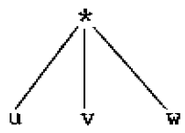
All newly generated OBJECT-SCHEMAs are processed by an IPL-V routine that does numeric simplification and removes unnecessary parentheses. Fig. 56.a is an example of an OBJECT-SCHEMA with unnecessary parentheses, which are removed in the equivalent OBJECT-SCHEMA in Fig. 56.b.

In some cases, two SETs that contain a different number of elements are matched. This fact is recognized by the match and a difference whose type is SET-SIZE is reported. (This is the only type of difference which is not a FEATURE and it is task independent.) The value of the difference is the unmatched elements of the SETs. Hence, it is necessary to consider only a subset of a SET. An example is GOAL 6 in Fig. 54. Consequently, the REDUCE-METHOD was generalized so that it could create a new object that consisted of a subset of a SET. When a difference is reduced on such an object, the REDUCE-METHOD substitutes the result for the subset in the SET. For example, in Fig. 54 OBJECT 17 is a subset of the set of factors in EXPRESSION-1. OBJECT 19 is derived from OBJECT 17 in an attempt to achieve GOAL 6 and is substituted in EXPRESSION-1, which results in OBJECT 20.

All of the other operators that are applied implicitly are given to GPS as immediate operators and are applied by the MATCH-DIFF-METHOD. Each of these is an IPL-V routine.

An alternative representation of the operators of this task is to represent them individually, instead of grouping them into two SETs of operators. Using such a representation, GPS would not select one whose input

(a)



(b)

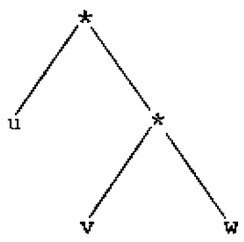


FIGURE 56. (a) is the tree structure representation of (u * (v * w)) and (b) is the tree structure representation of (u * v * w).

form was similar to an object but would try them in the order in which they appeared in the TABLE-OF-CONNECTIONS. This, of course, gives GPS less selectivity than the formulation in Fig. 51, particularly if the number of operators were increased.

It seems reasonable to group all integral forms together since they have the common function of removing an '$\int$'. And since there are a considerable number of integral forms and many more could be added, it seems reasonable to attempt to apply them in the order in which they are most likely to be applicable. Selecting the operators whose input forms are most similar to the object to which they are applied is synonomous with selecting the operator which seems to be most feasible to apply. Currently, the REDUCE-METHOD only selects desirable operators and does not evaluate them according to their feasibility. Perhaps the reduce method should also select operators which seem to be feasible as well as desirable[6].

Certain algebraic operators, such as

$$u*u = u^2$$

are not included in this formulation of integration. These would certainly be included in a more complete formulation. Some of the operators in Fig. 53 are equivalent. For example, OPERATOR 3 is the integral form of OPERATOR 8. Some operators are special cases of others, e.g., OPERATOR 5 is a special case of OPERATOR 1. A better formulation of the task would not have such redundancies.

---

[6]This is also proposed on page 181 in Newell, et al [29].

C. TOWER OF HANOI

In the Tower of Hanoi, which is a classical puzzle, there are three pegs and a number of disks, each of whose diameter is different from all of the others. Initially, all of the disks are stacked on the first peg in order of descending size, as illustrated in Fig. 57. The problem is to discover a sequence of moves which will transfer all of the disks to the third peg. Each move consists of removing the top disk on any peg and placing it on top of the disks on another peg, but never placing a disk on top of one smaller than itself.

GPS Formulation

A presentation of the four disk version of the Tower of Hanoi to GPS is given in Fig. 58. TOP-GOAL is the statement that the problem is to produce the DESIRED-OBJ from the INITIAL-OBJ, which represents the situation when all of the disks are on the first peg (PEG-1). (----- is ignored by the translator.) INITIAL-OBJ is an OBJECT-SCHEMA and is illustrated as a tree structure in Fig. 59. The only node with a local description is PEG-1. (All of the ATTRIBUTEs of the other nodes are UNDEFINED.) The presence of a disk on a peg is indicated by the value, YES, of the disk (which is an ATTRIBUTE) at the node which represents the peg. The absence of a disk on a peg is indicated by the ATTRIBUTE corresponding to the disk being UNDEFINED at the node which represents the peg. Disks are the only ATTRIBUTES in this task (OBJ-ATTRIB). The DESIRED-OBJ is an OBJECT-SCHEMA that represents the situation when all of the disks are on PEG-3.

The only operator in this task is MOVE-DISK which moves a disk from one peg to another peg, provided that it is legitimate. The operator, MOVE-DISK, contains four free variables:

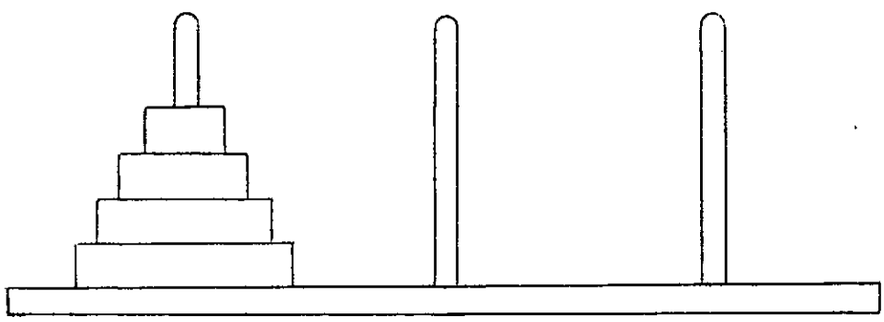


FIGURE 57. A "front view" of the initial situation of the Tower of Hanoi.

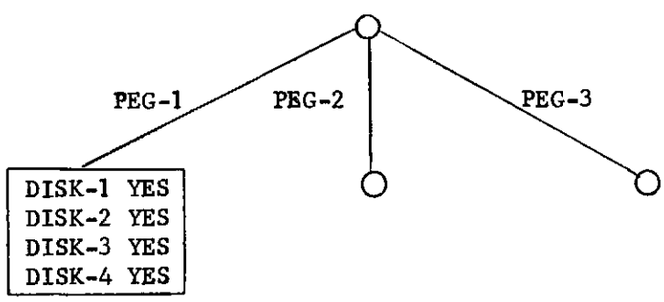


FIGURE 59. The tree structure representation of the INITIAL-OBJ in the Tower of Hanoi.

```
RENAME   (
   PEG-1 = FIRST
   PEG-2 = SECOND
   PEG-3 = THIRD
    )
DECLARE (
   DISK = ATTRIBUTE
   DISKS = SET
   DISK-1 = ATTRIBUTE
   DISK-2 = ATTRIBUTE
   DISK-3 = ATTRIBUTE
   DISK-4 = ATTRIBUTE
   D1.1 = FEATURE
   D1.2 = FEATURE
   D1.3 = FEATURE
   D2.1 = FEATURE
   D2.2 = FEATURE
   D2.3 = FEATURE
   D3.1 = FEATURE
   D3.2 = FEATURE
   D3.3 = FEATURE
   D4.1 = FEATURE
   D4.2 = FEATURE
   D4.3 = FEATURE
   DESIRED-OBJ = OBJECT-SCHEMA
   FROM-PEG = LOC-PROG
   INITIAL-OBJ = OBJECT-SCHEMA
```

Figure 58: The specification for GPS of the Tower of Hanoi.

```
    MOVE-DISK = MOVE-OPERATOR
    NONE = SET
    OTHER-PEG = LOC-PROB
    PEGS = SET
    SMALLER = ATTRIBUTE
    THE-FIRST = SET
    THE-FIRST-2 = SET
    THE-FIRST-3 = SET
    TO-PEG = LOC-PROB
    X = ATTRIBUTE
      )
TASK-STRUCTURES (
    TOP-GOAL = ( TRANSFORM THE INITIAL-OBJ INTO THE DESIRED-OBJ )
    INITIAL-OBJ = ( PEG-1 ( DISK-1 YES DISK-2 YES DISK-3 YES DISK-4 YES )
                    PEG-2 ( ----- )
                    PEG-3 ( ----- )    )
    OBJ-ATTRIB = ( DISK-1 DISK-2 DISK-3 DISK-4 )
    DESIRED-OBJ = ( PEG-1 ( ----- )
                    PEG-2 ( ----- )
                    PEG-3 ( DISK-1 YES DISK-2 YES DISK-3 YES DISK-4 YES ) )
    MOVE-DISK = ( CREATION-OPERATION
                  VAR-DOMAIN
                  1. THE TO-PEG IS AN EXCLUSIVE-MEMBER OF THE PEGS ,
                  2. THE FROM-PEG IS AN EXCLUSIVE-MEMBER OF THE PEGS ,
                  3. THE OTHER-PEG IS AN EXCLUSIVE-MEMBER OF THE PEGS ,
                  4. THE DISK IS IN-THE-SET OF DISKS ,
                  PRECISTS
```

Figure 58: (continued)

```
                    1. X ON THE OTHER-PEG IS DEFINED FOR-ALL X SMALLER
                       THAN THE PARTICULAR DISK .
                  MOVES
                    1. MOVE THE DISK ON THE FROM-PEG TO THE DISK ON THE TO-PEG .

DISKS = ( DISK-1 DISK-2 DISK-3 DISK-4 )
PEGS = ( PEG-1 PEG-2 PEG-3 )
DISK-1 = ( THE SMALLER ONES ARE NONE )
DISK-2 = ( THE SMALLER ONE IS THE-FIRST )
DISK-3 = ( THE SMALLER ONES ARE THE-FIRST-2 )
DISK-4 = ( THE SMALLER ONES ARE THE-FIRST-3 )
NONE = ( ----- )
THE-FIRST = ( DISK-1 )
THE-FIRST-2 = ( DISK-1 DISK-2 )
THE-FIRST-3 = ( DISK-1 DISK-2 DISK-3 )
D1.1 = ( DISK-1 ON PEG-1 , )
D1.2 = ( DISK-1 ON PEG-2 , )
D1.3 = ( DISK-1 ON PEG-3 , )
D2.1 = ( DISK-2 ON PEG-1 , )
D2.2 = ( DISK-2 ON PEG-2 , )
D2.3 = ( DISK-2 ON PEG-3 , )
D3.1 = ( DISK-3 ON PEG-1 , )
D3.2 = ( DISK-3 ON PEG-2 , )
D3.3 = ( DISK-3 ON PEG-3 , )
D4.1 = ( DISK-4 ON PEG-1 , )
D4.2 = ( DISK-4 ON PEG-2 , )
D4.3 = ( DISK-4 ON PEG-3 , )
COMPARE-OBJECTS = ( BASIC-MATCH )
```

Figure 58: (continued)

```
BASIC-MATCH = ( COMP-FEAT-LIST ( D1,1 D1,2 D1,3 D2,1 D2,2 D2,3 D3,1 D3,2
                                 D3,3 D4,1 D4,2 D4,3 ) )
DIFF-ORDERING = ( 1. ( D4,1 D4,2 D4,3 )
                  2. ( D3,1 D3,2 D3,3 )
                  3. ( D2,1 D2,2 D2,3 )
                  4. ( D1,1 D1,2 D1,3 ) )
TABLE-OF-CONNECTIONS = ( ( COMMON-DIFFERENCE MOVE-DISK ) )
LIST-OF-VARS = ( DISK FROM-PEG OTHER-PEG TO-PEG X )
  )
END
```

Figure 58: (continued)

a. FROM-PEG is the peg from which the disk is moved;
b. TO-PEG is the peg to which the disk is moved;
c. OTHER-PEG is neither the FROM-PEG nor the TO-PEG;
d. DISK is the disk which is moved.

The first three TESTs following VAR-DOMAIN insure that FROM-PEG, TO-PEG, and OTHER-PEG all stand for pegs and that no two of them stand for the same peg. The fourth TEST following VAR-DOMAIN is the statement that DISK stands for one of the four disks.

The formulation of MOVE-DISK is based on the fact that the top disk on a peg is also the smallest disk on the peg. Since this is true of the INITIAL OBJ, it is true of all other objects because a disk is never placed on a smaller disk. Thus, in order to move the DISK, it must be the smallest disk on the FROM-PEG as well as smaller than any disk on TO-PEG. Due to the conservation of disks, these two constraints can be restated as: All of the disks which are smaller than DISK must be on the OTHER-PEG. This is the meaning of the TEST following PRETESTS. If the disk, X, at the node, OTHER-PEG, is DEFINED (i.e., X is an ATTRIBUTE of the node that OTHER-PEG stands for), X is on the OTHER-PEG. The TEST following PRETESTS is true only if any disk, X, which is smaller than DISK is on the OTHER-PEG.

FOR-ALL is used as a replicator. That is, the TESTs following PRETESTS is actually several TESTs (the conjunction thereof). Each of these TESTs has the form,

X ON THE OTHER PEG is DEFINED,

and each has a different value for X, which is indicated, syntactically, by the fact that X follows FOR-ALL. The domain of X is the SET designated by the phrase (whose semantics are discussed below)

SMALLER THAN THE PARTICULAR DISK.

Under MOVES there is only one TRANSFORMATION, which has the effect of moving the DISK (which must be on the FROM-PEG) from the FROM-PEG to the TO-PEG.

DISKS is the SET of disks and PEGS is the SET of pegs both of which are used in the stating of the TESTs in VAR-DOMAIN in MOVE-DISK.

Although each disk is an ATTRIBUTE, (e.g., DISK-3), it is also a data structure. With each disk is associated one attribute-value pair: SMALLER is the ATTRIBUTE and its value is the SET of those disks which are smaller than the disk. For example, in DISK-3, the value of SMALLER is THE-FIRST-2 which is the set of two disks, DISK-1, DISK-2. This information, which defines the size of a disk relative to the other disks, is used in the TEST following PRETESTS in MOVE-DISK. When DISK has the value DISK-3, the phrase

SMALLER THAN THE PARTICULAR DISK

designates the value of the ATTRIBUTE, SMALLER, of DISK-3, which is THE-FIRST-2. SMALLER in this example is a FEATURE. PARTICULAR DISK indicates that this FEATURE does not refer to the input object of the operator but to the data structure whose name is the value of the variable DISK.

D1.1 through D4.3 in Fig. 58 are the types of differences in this task. Each is a FEATURE. COMPARE-OBJECTS indicates that the match should check for the type of differences listed after COMP-FEAT-LIST in BASIC-MATCH. The match only checks for differences at the TOP-NODE of objects.

DIFF-ORDERING is the statement that the larger the disk, the more difficult the difference. In DIFF-ORDERING, the difference types are divided into four categories. All of the difference types in a category pertain to a paticular disk and are considered equally difficult. The

categories are arranged according to the difficulty of their difference types which is determined by the size of the disk to which they pertain.

The TABLE-OF-CONNECTIONS indicates that MOVE-DISK is to be used to reduce any type of difference. This means that no selection power comes directly from the TABLE-OF-CONNECTIONS; instead it must come from the desirability-selection process for MOVE-OPERATORs.

LIST-OF-VAR is a list of symbols in Fig. 58 which are variables.

Behavior of GPS

The way that GPS arrives at the solution to the Tower of Hanoi is shown in Fig. 60. In attempting TOP-GOAL, GPS notices that DISK-4 should be on PEG-3 (i.e., the ATTRIBUTE, DISK-4 of the node named PEG-3 should have the value, YES) and creates GOAL 2 to reduce this difference. GPS also notices that the other three disks should be on PEG-3, but selects the difference whose type is D4.3 because it is the most difficult difference detected.

By referring to TABLE-OF-CONNECTIONS, GPS selects MOVE-DISK to be applied to INITIAL-OBJ. The desirability-selection process finds that MOVE-DISK performs a desirable function if the variables, TO-PEG and DISK, have the respective values PEG-3 and DISK-4. The application of MOVE-DISK with these two variables so specified is attempted by creating GOAL 3. After specifying the remaining variables, FROM-PEG and OTHER-PEG, to be PEG-1 and PEG-2 respectively, GPS detects that the operator is infeasible because DISK-1, DISK-2, and DISK-3 are not on PEG-2. GOAL 4 is created to reduce the difference that DISK-3 is not on PEG-2 which is the most difficult difference detected in attempting to apply the operator. However, it is less difficult than the difference for which GOAL 2 was created; otherwise GOAL 4 would have been considered undesirable. The only other legitimate variable

```
1 TOP-GOAL TRANSFORM INITIAL-OBJ INTO DESIRED-OBJ      (SUBGOAL OF NONE)

  2 GOAL 2 REDUCE D4,3 ON INITIAL-OBJ      (SUBGOAL OF TOP-GOAL)

    3 GOAL 3 APPLY MOVE-DISK WITH TO-PEG = PEG-3, DISK = DISK-4, TO INITIAL-OBJ      (SUBGOAL OF 2)
                SET: FROM-PEG = PEG-1, OTHER-PEG = PEG-2

      4 GOAL 4 REDUCE D3,2 ON INITIAL-OBJ      (SUBGOAL OF 3)

        5 GOAL 5 APPLY MOVE-DISK WITH TO-PEG = PEG-2, DISK = DISK-3, TO INITIAL-OBJ      (SUBGOAL OF 4)
                SET: FROM-PEG = PEG-1, OTHER-PEG = PEG-3

          6 GOAL 6 REDUCE D2,3 ON INITIAL-OBJ      (SUBGOAL OF 5)

            7 GOAL 7 APPLY MOVE-DISK WITH TO-PEG = PEG-3, DISK = DISK-2, TO INITIAL-OBJ      (SUBGOAL OF 6)
                SET: FROM-PEG = PEG-1, OTHER-PEG = PEG-2

              8 GOAL 8 REDUCE D1,2 ON INITIAL-OBJ      (SUBGOAL OF 7)

                9 GOAL 9 APPLY MOVE-DISK WITH TO-PEG = PEG-2, DISK = DISK-1, TO INITIAL-OBJ      (SUBGOAL OF 8)
                    SET: FROM-PEG = PEG-1, OTHER-PEG = PEG-3
                    OBJECT 5: (PEG-1(DISK-2 YES DISK-3 YES DISK-4 YES) PEG-2(DISK-1 YES) PEG-3(-----
                              ))

              8 GOAL 10 APPLY MOVE-DISK WITH TO-PEG = PEG-3, DISK = DISK-2, TO 5      (SUBGOAL OF 7)
                    SET: FROM-PEG = PEG-1, OTHER-PEG = PEG-2
                    OBJECT 6: (PEG-1(DISK-3 YES DISK-4 YES) PEG-2(DISK-1 YES) PEG-3(DISK-2 YES))

            6 GOAL 11 APPLY MOVE-DISK WITH TO-PEG = PEG-2, DISK = DISK-3, TO 6      (SUBGOAL OF 5)
                    SET: FROM-PEG = PEG-1, OTHER-PEG = PEG-3

              7 GOAL 12 REDUCE D1,3 ON 6      (SUBGOAL OF 11)

                8 GOAL 13 APPLY MOVE-DISK WITH TO-PEG = PEG-3, DISK = DISK-1, TO 6      (SUBGOAL OF 12)
                    SET: FROM-PEG = PEG-2, OTHER-PEG = PEG-1
                    OBJECT 7: (PEG-1(DISK-3 YES DISK-4 YES) PEG-2(-----) PEG-3(DISK-1 YES DISK-2 YES
                              ))

              7 GOAL 14 APPLY MOVE-DISK WITH TO-PEG = PEG-2, DISK = DISK-3, TO 7      (SUBGOAL OF 11)
                    SET: FROM-PEG = PEG-1, OTHER-PEG = PEG-3
                    OBJECT 8: (PEG-1(DISK-4 YES) PEG-2(DISK-3 YES) PEG-3(DISK-1 YES DISK-2 YES))
```

Figure 60: The performance of GPS on the Tower of Hanoi.

```
    4 GOAL 15 APPLY MOVE-DISK WITH TO-PEG = PEG-3, DISK = DISK-4, TO 8      (SUBGOAL OF 3)
                SET:  FROM-PEG = PEG-1, OTHER-PEG = PEG-2

      5 GOAL 16 REDUCE D2,2 ON 8      (SUBGOAL OF 15)

        6 GOAL 17 APPLY MOVE-DISK WITH TO-PEG = PEG-2, DISK = DISK-2, TO 8      (SUBGOAL OF 16)
                SET:  FROM-PEG = PEG-3, OTHER-PEG = PEG-1

          7 GOAL 18 REDUCE D1,1 ON 8      (SUBGOAL OF 17)

            8 GOAL 19 APPLY MOVE-DISK WITH TO-PEG = PEG-1, DISK = DISK-1, TO 8      (SUBGOAL OF 18)
                SET:  FROM-PEG = PEG-3, OTHER-PEG = PEG-2
                OBJECT 9: (PEG-1(DISK-1 YES DISK-4 YES) PEG-2(DISK-3 YES) PEG-3(DISK-2 YES))

          7 GOAL 20 APPLY MOVE-DISK WITH TO-PEG = PEG-2, DISK = DISK-2, TO 9      (SUBGOAL OF 17)
                SET:  FROM-PEG = PEG-3, OTHER-PEG = PEG-1
                OBJECT 10: (PEG-1(DISK-1 YES DISK-4 YES) PEG-2(DISK-2 YES DISK-3 YES) PEG-3(-----
                           ))

      5 GOAL 21 APPLY MOVE-DISK WITH TO-PEG = PEG-3, DISK = DISK-4, TO 10      (SUBGOAL OF 15)
                SET:  FROM-PEG = PEG-1, OTHER-PEG = PEG-2

        6 GOAL 22 REDUCE D1,2 ON 10      (SUBGOAL OF 21)

          7 GOAL 23 APPLY MOVE-DISK WITH TO-PEG = PEG-2, DISK = DISK-1, TO 10      (SUBGOAL OF 22)
                SET:  FROM-PEG = PEG-1, OTHER-PEG = PEG-3
                OBJECT 11: (PEG-1(DISK-4 YES) PEG-2(DISK-1 YES DISK-2 YES DISK-3 YES) PEG-3(-----
                           ))

        6 GOAL 24 APPLY MOVE-DISK WITH TO-PEG = PEG-3, DISK = DISK-4, TO 11      (SUBGOAL OF 21)
                SET:  FROM-PEG = PEG-1, OTHER-PEG = PEG-2
                OBJECT 12: (PEG-1(-----) PEG-2(DISK-1 YES DISK-2 YES DISK-3 YES) PEG-3(DISK-4 YES
                           ))

  2 GOAL 25 TRANSFORM 12 INTO DESIRED-OBJ      (SUBGOAL OF TOP-GOAL)

    3 GOAL 26 REDUCE D3,3 ON 12      (SUBGOAL OF 25)

      4 GOAL 27 APPLY MOVE-DISK WITH TO-PEG = PEG-3, DISK = DISK-3, TO 12      (SUBGOAL OF 26)
                SET:  FROM-PEG = PEG-2, OTHER-PEG = PEG-1

        5 GOAL 28 REDUCE D2,1 ON 12      (SUBGOAL OF 27)

          6 GOAL 29 APPLY MOVE-DISK WITH TO-PEG = PEG-1, DISK = DISK-2, TO 12      (SUBGOAL OF 28)
```

Figure 60: (continued)

```
            SET: FROM-PEG = PEG-2, OTHER-PEG = PEG-3

      7 GOAL 31 REDUCE D1,3 ON 12      (SUBGOAL OF 29)

        8 GOAL 31 APPLY MOVE-DISK WITH TO-PEG = PEG-3, DISK = DISK-1, TO 12      (SUBGOAL OF 30)
            SET: FROM-PEG = PEG-2, OTHER-PEG = PEG-1
            OBJECT 13: (PEG-1(------) PEG-2(DISK-2 YES DISK-3 YES) PEG-3(DISK-1 YES DISK-4 YES
                       ))

      7 GOAL 32 APPLY MOVE-DISK WITH TO-PEG = PEG-1, DISK = DISK-2, TO 13      (SUBGOAL OF 29)
            SET: FROM-PEG = PEG-2, OTHER-PEG = PEG-3
            OBJECT 14: (PEG-1(DISK-2 YES) PEG-2(DISK-3 YES) PEG-3(DISK-1 YES DISK-4 YES))

    6 GOAL 33 APPLY MOVE-DISK WITH TO-PEG = PEG-3, DISK = DISK-3, TO 14      (SUBGOAL OF 27)
            SET: FROM-PEG = PEG-2, OTHER-PEG = PEG-1

      6 GOAL 34 REDUCE D1,1 ON 14      (SUBGOAL OF 33)

        7 GOAL 35 APPLY MOVE-DISK WITH TO-PEG = PEG-1, DISK = DISK-1, TO 14      (SUBGOAL OF 34)
            SET: FROM-PEG = PEG-3, OTHER-PEG = PEG-2
            OBJECT 15: (PEG-1(DISK-1 YES DISK-2 YES) PEG-2(DISK-3 YES) PEG-3(DISK-4 YES))

      6 GOAL 36 APPLY MOVE-DISK WITH TO-PEG = PEG-3, DISK = DISK-3, TO 15      (SUBGOAL OF 33)
            SET: FROM-PEG = PEG-2, OTHER-PEG = PEG-1
            OBJECT 16: (PEG-1(DISK-1 YES DISK-2 YES) PEG-2(------) PEG-3(DISK-3 YES DISK-4 YES
                       ))

3  GOAL 37 TRANSFORM 16 INTO DESIRED-OBJ      (SUBGOAL OF 25)

  4 GOAL 38 REDUCE D2,3 ON 16      (SUBGOAL OF 37)

    5 GOAL 39 APPLY MOVE-DISK WITH TO-PEG = PEG-3, DISK = DISK-2, TO 16      (SUBGOAL OF 38)
            SET: FROM-PEG = PEG-1, OTHER-PEG = PEG-2

    .
      6 GOAL 41 REDUCE D1,2 ON 16      (SUBGOAL OF 39)

        7 GOAL 41 APPLY MOVE-DISK WITH TO-PEG = PEG-2, DISK = DISK-1, TO 16      (SUBGOAL OF 40)
            SET: FROM-PEG = PEG-1, OTHER-PEG = PEG-3
            OBJECT 17: (PEG-1(DISK-2 YES) PEG-2(DISK-1 YES) PEG-3(DISK-3 YES DISK-4 YES))

      6 GOAL 42 APPLY MOVE-DISK WITH TO-PEG = PEG-3, DISK = DISK-2, TO 17      (SUBGOAL OF 39)
            SET: FROM-PEG = PEG-1, OTHER-PEG = PEG-2
            OBJECT 18: (PEG-1(------) PEG-2(DISK-1 YES) PEG-3(DISK-2 YES DISK-3 YES DISK-4 YES
                       ))
    .
  4 GOAL 43 TRANSFORM 18 INTO DESIRED-OBJ      (SUBGOAL OF 37)
```

Figure 60: (continued)

```
5 GOAL 44 REDUCE D1.3 ON 18      (SUBGOAL OF 43)

  6 GOAL 45 APPLY MOVE-DISK WITH TO-PEG = PEG-3, DISK = DISK-1, TO 18      (SUBGOAL OF 44)
            SET: FROM-PEG = PEG-2, OTHER-PEG = PEG-1
            OBJECT 19: (PEG-1(-----) PEG-2(-----) PEG-3(DISK-1 YES DISK-2 YES DISK-3 YES DISK-4
                  YES))

5 GOAL 46 TRANSFORM 19 INTO DESIRED-OBJ      (SUBGOAL OF 43)
SUCCESS
```

Figure 60: (continued)

specification for FROM-PEG and TO-PEG is PEG-2 and PEG-1, respectively, which would give rise to the difference that DISK-4 is not on PEG-2. Since this difference is as difficult as the difference for which GOAL 2 was created, GPS considers this variable specification to be undesirable.

GPS attempts to achieve GOAL 4 by moving DISK-3 to PEG-2 for which GOAL 5 was created. Since DISK-2 is not on PEG-3 (GOAL 6), GOAL 7 is created in an attempt to move DISK-2 to PEG-3. In order to achieve GOAL 7, DISK-1 is moved to PEG-2 (GOAL 8 and GOAL 9) which results in the new object, OBJECT 5. DISK-2 was moved to PEG-3 in OBJECT 5 (GOAL 10) and OBJECT 6 is produced.

In reattempting GOAL 4, GOAL 11 is created to move DISK-3 to PEG-2 in OBJECT 6. After moving DISK-1 from PEG-2 to PEG-3, (GOAL 12 and GOAL 13) DISK-3 is moved to PEG-2 (GOAL 14) and GOAL 4 is achieved. The first fourteen goals are typical of the behavior of GPS on this task, and success is achieved at GOAL 46.

Discussion

In moving a disk, two constraints, which are usually stated somewhat independently, must be satisfied. Only the top disk on a peg can be moved and it must be smaller than the top disk on the peg to which it is being moved. In MOVE-DISK these two constraints are replaced by the single TEST of PRETESTS. Combining the two constraints significantly changes the problem formulation even though both formulations are isomorphic.

The more usual formulation of MOVE-DISK would have the following PRETESTS:

1. X ON THE TO-PEG IS UNDEFINED FOR-ALL X
   SMALLER THAN THE-PARTICULAR DISK.

2. X ON THE FROM-PEG IS UNDEFINED FOR-ALL X SMALLER THAN THE-PARTICULAR DISK.

With this formulation, GPS would find the task considerably more difficult[7]. Suppose, for example, that GPS wanted to move DISK-2 to PEG-3 and that DISK-1 and DISK-2 were on PEG-1 (GOAL 7 in Fig. 60). PEG-1 would be used for FROM-PEG and GPS would only find one difference,

DISK-1 should not be on PEG-1.

This difference contains no information about where DISK-1 should be moved, and it might be moved to PEG-3 in an attempt to reduce the difference. However, using the formulation in Fig. 58, GPS detects the difference,

DISK-1 should be on PEG-2,

(GOAL 8) because OTHER-PEG must equal PEG-2 in order for GPS to consider the variable specification desirable. In reducing this difference, GPS considers it desirable to move DISK-1 to PEG-3 (GOAL 9) but undesirable to move DISK-1 to PEG-2.

An outstanding feature of GPS's behavior on the Tower of Hanoi is that GPS never makes a mistake. That is, GPS always chooses to reduce a difference which leads to the selection of the best operator. As noted above, this is not the case when the PRETESTS in MOVE-DISK consists of two separate constraints. Thus, formulating the PRETESTS as a single constraint is necessary for GPS to always select the correct operator. Another factor which allows GPS to select the correct operators is that the types of differences that GPS

7In attempting the four disk version of the Tower of Hanoi, GPS would exhaust memory before finding a solution. GPS could probably find the solution to the three disk version.

detects and their relative difficulty are in some sense optimal. For many tasks a good set of differences and a good DIFF-ORDERING are difficult to obtain.

Gaku [9], another problem solver which solved the Tower of Hanoi, approaches the task quite differently than GPS. First, Gaku solves the "three disk" Tower of Hanoi by trial and error. Then, the solution to the four disk task is obtained by generalizing the solution to the three disk task; the five disk solution is obtained by generalizing the four disk solution, etc. Hence, Gaku solves this task by searching for a solution in a space of solutions whereas GPS searches for the searches for the desired situation in a space of situations.

D. PROVING THEOREMS EXPRESSED IN THE FIRST-ORDER PREDICATE CALCULUS

The first-order predicate calculus, sometimes called the first-order predicate logic or quantification theory, is a formal language system that has received much attention in attempts to construct theorem proving programs. (For a representative sample, see Davis and Putnam [9]; Friedman [14]; Gilmore [17]; Robinson [50,51]; Wang [62,63]; Wos, et al [67].)

The primitive symbols of this logic are:

a. parentheses--(,)
b. logical connectives--$\supset$, &, V, $\equiv$,
c. predicate letters--P, Q, R, ...
d. individual names--a, b, c, ...
e. variables--u, v, w, ...
f. quantifiers--$\exists$, $\forall$.

Parentheses serve their normal purpose of delimiting groups of symbols. The logical connectives stand for negation, conjunction, disjunction, implication, and equivalence, respectively. An atomic formula is a predicate letter

followed by a (, followed by a string of several individual names or variables separated by commas, followed by a). The intent of the quantifiers can best be given by example: $(\exists u)\ P(u)$ is true if and only if there exists some individual name, α, such that $P(\alpha)$ is true. $(\forall u)\ P(u)$ is true if and only if for every possible individual name, α, $P(\alpha)$ is true.

Formulae F_1, F_2, F_3,... of the calculus can be recursively defined as one of the following (where F_i and F_j are arbitrarily formulae):

a. an atomic formula
b. F_i
c. $(F_i \ \& \ F_j)$
d. $(F_i \ V \ F_j)$
e. $(F_i \supset F_j)$
f. $(F_i \equiv F_j)$
g. a quantified formula such as $(\forall u)\ F_i$ or $(\exists u)\ F_i$

A formula may be either true or false depending on the interpretations given to the predicate letters and individual names that occur in it. For example, $(\exists u)\ (P(b,u)\ \&\ Q(u))$ is true when $P(b,u)$ is understood as 'b is the brother of u', $Q(u)$ is understood as 'u is tall', and 'b' is the name of someone within the scope of u with a tall brother.

If a ground formula is defined as either an atomic formula or the negation of an atomic formula, then an interpretation can be defined as a set, I, of ground formulae with the following properties:

a. No atomic formula and its negation are members of I (I is consistent).
b. For every possible atomic formula that can be formed from the individual names and predicate letters occurring in the formula in I, either the atomic formula or its negation is a member of I (I is complete).

A formula is satisfiable if there exists an interpretation for which it is true; otherwise, it is unsatisfiable. A formula is valid, if it is true for

all possible interpretations. A formula is a theorem if an only if the formula is valid. It follows that the negation of a theorem must necessarily be unsatisfiable.

Predicate Calculus Theorem Provers

Although several different approaches have been taken in constructing theorem proving programs, we will only consider the one which has received the most attention. The theoretical justification of this approach can be found in Davis and Putnam [9]; Gilmore [17]; Robinson [51]. These theorem provers do not process formulae in the calculus directly, but first reduce the statement of the theorem to a cannonical form in order to simplify the theorem proving process. In this cannonical form, an atomic formula can have functions, f, g, h,...of variables within the scope of a predicate,--e.g., P(f(a))--as well as individual names and variables. (The intent of these functions will be discussed later.) A literal is defined as an atomic formula or the negation of an atomic formula. The disjunction of a finite set of literals is called a clause, and the empty clause is denoted by $\square$ The cannonical form of a theorem is the conjunction of a finite set of clauses. Instead of proving that a theorem is valid, the theorem provers prove that the negation of the theorem is unsatisfiable; i.e., a contradiction can be obtained.

Neither the logical connectives $\supset$, $\equiv$, nor the quantifiers, $\forall$ and $\exists$, appear in the cannonical form. However, there is no loss of generality in the use of this cannonical form because any formula (theorem) in the calculus can be reduced to this cannonical form. Implication and equivalence can be defined in terms of conjunction, negation, and disjunction, and can be removed by repeated application of these definitions. $\forall$ does not appear in

the cannonical form, because by convention all variables are universally quantified. All occurrences of ∃ are removed by replacing all of the variables quantified by ∃ with functions of the universally quantified variables on which they depend. For example, the cannonical form of the formula (∀u) (∃v) (∀w) P(u,v,w), is P(u,f(u),w). The variables u and w are universally quantified by convention. In order for the formula to be true, there must exist a value of v which will make the formula true for every possible value of w. But this value of v does not have to be the same for every possible value of u. It can be a function of u; hence, v can be replaced by a function of u, i.e., f(u). On the other hand, in the formula, (∃v) (∀u) (∀w) P(u, v, w), whose cannonical form is P(u,f,w), the value of v must be independent of both u and w; i.e., f must be a constant.

These theorem provers use only a single inference principle called the resolution principle (Robinson [51]; Wos [67]). From any two clauses, C and D, a resolvent of C and D can (possibly) be inferred, which has the following properties:

a. L is a literal in C and M is a literal in D.
b. The resolvent is a new clause consisting of all of the literals in C and D with the exception of L and M.
c. Either L or M but not both contain a ~.
d. Except for the ~, L and M are identical or can be made identical by substituting for variables, functions, individual names, or other variables.
e. Any substitutions which must be made in L and M in order to make them identical are also made in the resolvent.

If there are no L and M which fulfill the above conditions, C and D will have no resolvents; and if L and M are not uniquely specified by the above conditions, C and D will have more than one resolvent.

If S is any set of clauses, the nth resolution of S, denoted by $\mathfrak{R}^n(S)$, is defined as:

a. $\mathfrak{R}^1(S)$ is all members of S together with all resolvents of all pairs of members of S.
b. For $n \geq 1$, $\mathfrak{R}^{n+1}(S) = \mathfrak{R}(\mathfrak{R}^n(S))$

Then the resolution principle is

Resolution Theorem: If S is any finite set of clauses, then S is unsatisfiable if and only if $\mathfrak{R}^n(S)$ contains $\square$, for some $n \geq 1$.

Theorem provers based on the Resolution Theorem attempt to prove the unsatisfiability of a set of clauses by inferring $\square$. The proof of the unsatisfiability of a set, S, of clauses (proof of a theorem) consists of a sequence of clauses, each of which is either a member of S or the resolvent of two earlier clauses in the sequence. The final member of the sequence is $\square$. No such theorem prover can guarantee that it can either prove or disprove an arbitrary theorem in a finite amount of time (and computer memory) because the proof sequence can be, according to the Resolution Theorem, arbitrarily long.

A simple example of the canonical form of a theorem is the conjunction of the three clauses,

$$Q(b) \qquad (1)$$
$$\sim Q(u) \vee P(f(u)) \qquad (2)$$

and

$$\sim P(v) \qquad (3)$$

This theorem can be proved by first inferring the resolvent,

$$P(f(b)), \qquad (4)$$

of (1) and (2). In forming this resolvent, the two literals containing a Q cancel out by substituting b for u. Then, $\square$, which is a resolvent of (3) and (4) can be inferred.

GPS Formulation

GPS can prove theorems expressed in the cannonical form of the first-order predicate calculus, described above. Its proof techniques are very similar to the theorem provers we have just discussed in that it searches for □ by producing the resolvents of clauses.

Fig. 61 is the formulation for GPS of the task of proving the first-order predicate calculus theorem[8],

$$(\exists u)(\exists y)(\forall z)((P(u,y)\ (P(y,z)\ \&\ P(z,z)))\ \&\ ((P(u,y)\ \&\ Q(u,y))\ (Q(u,z)\ \&\ Q(z,z)))).$$

This theorem has a long history as a sample problem for theorem provers (Davis and Putnam [9]; Gilmore [17]; Robinson [50]). The early theorem provers found it very difficult, whereas it is trivial for the most recent programs.

TOP-GOAL in Fig. 61 is a statement of the problem. The INITIAL-OBJ is the statement of the theorem in cannonical form. The negation of the theorem, after removing the quantifiers and implication, is

$$\sim((\sim P(u,y) \vee (P(y,f(u,y))\ \&\ P(f(u,y),\ f(u,y))))\ \&\ (\sim(P(u,y)\ \&\ Q(u,y))\ \vee\ (Q(u,f(u,y))\ \&\ Q(f(u,y),\ f(u,y))))).$$

The INITIAL-OBJ is this expression written as a conjunction of the clauses, STATEMENT-1, STATEMENT-2, and STATEMENT-3. In the clauses in Fig. 61, all lower case letters are replaced by capital letters; ~ is replaced by -. There must be a space between all symbols in the clauses in Fig. 61 because "space" is the delimiter of words in the external representation of GPS.

Each clause is an OBJECT-SCHEMA, expressed as a linear string of symbols (translated in the LIST mode), and is converted into a tree structure, internally. This conversion routine, designed to process the objects and

[8]The source of this formula is #5 on page 265 in Church [7]. The notation used by Church is different than the notation used here.

```
RENAME    (
   AND = ,
   )
DECLARE      (
   F = UNARY-CONNECTIVE
   MAIN = FEATURE
   P = UNARY-CONNECTIVE
   Q = UNARY-CONNECTIVE
   V = N-ARY-CONNECTIVE
   V... = N-ARY-CONNECTIVE
   , = N-ARY-CONNECTIVE
   - = UNARY-CONNECTIVE
   )
LIST      (
   INITIAL-OBJ = ( STATEMENT-1 AND STATEMENT-2 AND STATEMENT-3 )
   STATEMENT-1 = ( P ( U , Y ) )
   STATEMENT-2 = ( - P ( Y , F ( U , Y ) ) V - P ( F ( U , Y ) , F ( U , Y ) )
                 V Q ( U , Y ) )
   STATEMENT-3 = ( - P ( Y , F ( U , Y ) ) V - P ( F ( U , Y ) , F ( U , Y ) )
                 V - Q ( U , F ( U , Y ) ) V - Q ( F ( U , Y ) , F ( U ,
                 Y ) ) )
   DESIRED-OBJ = ( FALSE )
   OPR-1 = ( ( B AND - B ) YIELDS FALSE )
   OPR-2 = ( ( ( B V... C ) AND - B ) YIELDS C )
   OPR-3 = ( ( ( B V... C ) AND ( - B V... D ) ) YIELDS ( C V D ) )
     )
TASK-STRUCTURES      (
```

Figure 61: The specification for GPS of the task of proving a theorem expressed in the predicate calculus.

```
TOP-GOAL = ( TRANSFORM THE INITIAL-OBJ INTO THE DESIRED-OBJ . )
MAIN = ( SYMBOL )
BASIC-MATCH = ( COMP-FEAT-LIST ( MAIN ) )
COMP-OBJECTS = ( BASIC-MATCH )
DIFF-ORDERING = ( MAIN )
TABLE-OF-CONNECTIONS = ( ( MAIN OPR-1 OPR-2 OPR-3 ) )
LIST-OF-OPR = ( OPR-1 OPR-2 OPR-3 )
LIST-OF-VAR = ( A C B Y D )
  )
END
```

Figure 61: (continued)

the operators of mathematical tasks, assumes that each node of OBJECT-SCHEMAs has two ATTRIBUTEs--SIGN and SYMBOL. Fig. 62 shows the tree structure representation of STATEMENT-1 and STATEMENT-3. The ATTRIBUTEs of the nodes are implied.

INITIAL-OBJ is a SET of OBJECT-SCHEMAs. (AND is the new name for $\wedge$ in this task; see RENAME.) GPS considers it to be the source of derivation of STATEMENT-1, STATEMENT-2, and STATEMENT-3 and all objects derived from them will also have INITIAL-OBJ as their source of derivation. Thus, INITIAL-OBJ is a set which grows as new objects are generated during problem-solving.

The DESIRED-OBJ, which represents $\square$, consists of a single node which has the value NULL for the ATTRIBUTE, SYMBOL.

This task has three operators, OPR-1, OPR-2, and OPR-3. Each operator has as an input two OBJECT-SCHEMAs separated by AND in Fig. 61. For example, the input to OPR-2 is two OBJECT-SCHEMAs; one has the form

B V... C,

and the form of the other is

-B,

where B and C are variables.

The OBJECT-SCHEMA,

B V... C,

represents a clause which is the disjunction of the literal, B, and other literals. C represents the disjunction of all of the literals in the OBJECT-SCHEMA except B. The reason for this representation will be discussed below.

COMP-OBJECTS and BASIC-MATCH indicate that the only type of difference to be detected in matching two OBJECT-SCHEMAs is MAIN and that it should only be checked for at the TOP-NODE of the OBJECT-SCHEMAs. This simple match is

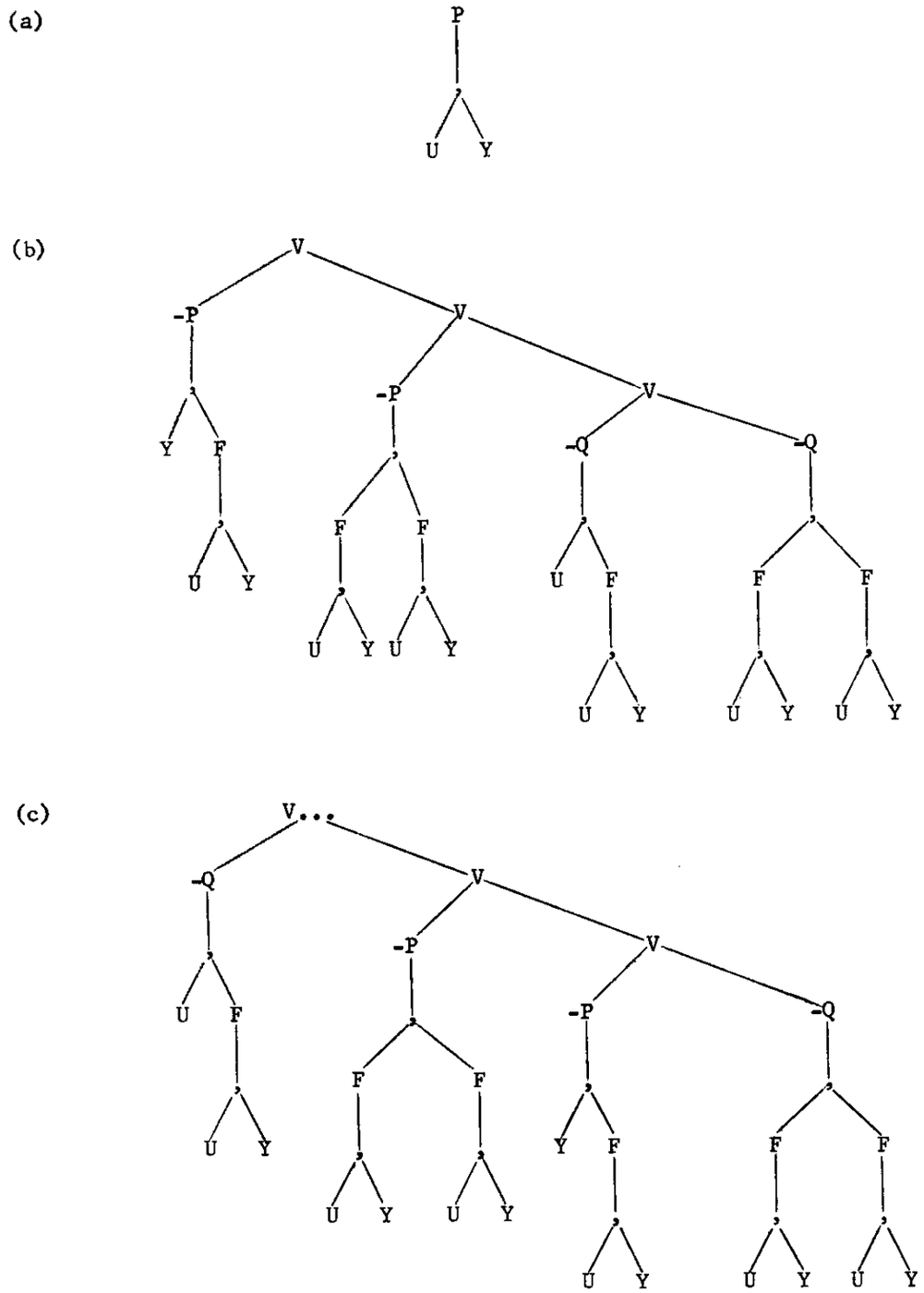


FIGURE 62. Tree structure representation of three predicate calculus objects. Only the values of the ATTRIBUTEs are shown.

sufficient for this task because matching two OBJECT-SCHEMAs is only done in the TRANSFORM-METHOD and DESIRED-OBJ is always one of the two. NULL only appears in an object representing □ . Thus only the value of the ATTRIBUTE, SYMBOL, of the TOP-NODE is checked; the rest of the OBJECT-SCHEMAs is ignored.

DIFF-ORDERING is only a formality for this task because there is only one type of difference. TABLE-OF-CONNECTIONS is also a formality for this task. It provides no information about the desirability of the operators because, according to it, all of the operators have the capability of reducing the only type of difference--MAIN.

LIST-OF-OPR indicates that the three operators need to be converted into their internal representation after being translated. The variables used in this task are given in LIST-OF-VAR.

Included in the presentation of this task are three criteria for selecting members from a SET of OBJECT-SCHEMAs. Since there is no provision in the external representation for expressing these criteria, which are used in SELECT GOALs, they are given to GPS as IPL structures and do not appear in Fig. 61. SMALLEST is a criterion (e.g., GOAL 2 in Fig. 63) which assigns the highest priority to those OBJECT-SCHEMAs (clauses) with the smallest number of literals. LIT-SIGN, which is another selection criterion, is satisfied only by those OBJECT-SCHEMAs which contain a literal whose SIGN and predicate letter are the same as the SIGN and predicate letter of the first literal in the criterion object. In addition, if the criterion object only contains a single literal, then LIT-SIGN requires that the OBJECT-SCHEMAs selected must not contain more than one literal. An OBJECT-SCHEMA selected according to the NO-LIT criterion must be a single literal if the criterion

```
1 TOP-GOAL TRANSFORM INITIAL-OBJ INTO DESIRED-OBJ      (SUBGOAL OF NONE)

  2 GOAL 2 SELECT FROM INITIAL-OBJ A/C SMALLEST OF DESIRED-OBJ      (SUBGOAL OF TOP-GOAL)
              STATEMENT-1 SELECTED

  2 GOAL 3 TRANSFORM STATEMENT-1 INTO DESIRED-OBJ      (SUBGOAL OF TOP-GOAL)

    3 GOAL 4 REDUCE MAIN ON STATEMENT-1      (SUBGOAL OF 3)

      4 GOAL 5 APPLY OPR-1 TO STATEMENT-1      (SUBGOAL OF 4)

        5 GOAL 6 SELECT FROM  CONDITION OPR-1 A/C NO.-LIT OF STATEMENT-1      (SUBGOAL OF  LEFT  CONDITION 5)
                  CONDITION  RIGHT SELECTED

        5 GOAL 7 APPLY  LEFT  CONDITION OPR-1 TO STATEMENT-1      (SUBGOAL OF 5)
                  OPERATOR 10:  ((P(J, Y) AND -P(U, Y)) YIELDS NULL)

        5 GOAL 8 SELECT FROM INITIAL-OBJ A/C LIT-SIGN OF  RIGHT  CONDITION 10      (SUBGOAL OF 5)
                  NONE SELECTED

      4 GOAL 9 APPLY OPR-2 TO STATEMENT-1      (SUBGOAL OF 4)

        5 GOAL 10 SELECT FROM  CONDITION OPR-2 A/C NO.-LIT OF STATEMENT-1      (SUBGOAL OF  RIGHT  CONDITION 9)
                  CONDITION  RIGHT SELECTED

        5 GOAL 11 APPLY  RIGHT  CONDITION OPR-2 TO STATEMENT-1      (SUBGOAL OF 9)
                  OPERATOR 11:  (((-P(U, Y) V... C) AND P(U, Y)) YIELDS C)

        5 GOAL 12 SELECT FROM INITIAL-OBJ A/C LIT-SIGN OF  LEFT  CONDITION 11      (SUBGOAL OF 9)
                  STATEMENT-3 SELECTED

        5 GOAL 13 APPLY  LEFT  CONDITION 11 TO STATEMENT-2      (SUBGOAL OF 9)
                  OBJECT 12: (-P(F(U, Y), F(U, Y)) V... Q(U, Y))

    3 GOAL 14 TRANSFORM 12 INTO DESIRED-OBJ      (SUBGOAL OF 3)

      4 GOAL 15 REDUCE MAIN ON 12      (SUBGOAL OF 14)

        5 GOAL 16 APPLY OPR-1 TO 12      (SUBGOAL OF 15)

          6 GOAL 17 SELECT FROM  CONDITION OPR-1 A/C NO.-LIT OF 12      (SUBGOAL OF 16)
                    NONE SELECTED
```

Figure 63: The performance of GPS on the task in Fig. 61.

```
5 GOAL 18 APPLY OPR-2 TO 12      (SUBGOAL OF 15)

  6 GOAL 19 SELECT FROM  CONDITION OPR-2 A/C NO.-LIT OF 12      (SUBGOAL OF  LEFT  CONDITION 18)
            CONDITION  LEFT SELECTED

  6 GOAL 20 APPLY  LEFT  CONDITION OPR-2 TO 12      (SUBGOAL OF 18)
            OPERATOR 13: (((~P(F(U, Y), F(U, Y)) V... Q(U, Y)) AND P(F(U, Y), F(U, Y))) YIELDS
                         Q(U, Y))

  6 GOAL 21 SELECT FROM INITIAL-OBJ A/C LIT-SIGN OF  RIGHT  CONDITION 13      (SUBGOAL OF 18)
            STATEMENT-1 SELECTED

  6 GOAL 22 APPLY  RIGHT  CONDITION 13 TO STATEMENT-1      (SUBGOAL OF 18)
            OBJECT 14: Q(U, Y)

4 GOAL  23 TRANSFORM 14 INTO DESIRED-OBJ      (SUBGOAL OF 14)

  5 GOAL 24 REDUCE MAIN ON 14      (SUBGOAL OF 23)

    6 GOAL 25 APPLY OPR-1 TO 14      (SUBGOAL OF 24)

      7 GOAL 26 SELECT FROM  CONDITION OPR-1 A/C NO.-LIT OF 14      (SUBGOAL OF  LEFT  CONDITION 25)
                CONDITION  RIGHT SELECTED

      7 GOAL 27 APPLY  LEFT  CONDITION OPR-1 TO 14      (SUBGOAL OF 25)
                OPERATOR 15: ((Q(U, Y) AND ~Q(U, Y)) YIELDS NULL)

      7 GOAL 28 SELECT FROM INITIAL-OBJ A/C LIT-SIGN OF  RIGHT  CONDITION 15      (SUBGOAL OF 25)
                NONE SELECTED

    6 GOAL 29 APPLY OPR-2 TO 14      (SUBGOAL OF 24)

      7 GOAL 30 SELECT FROM  CONDITION OPR-2 A/C NO.-LIT OF 14      (SUBGOAL OF  RIGHT  CONDITION 29)
                CONDITION  RIGHT SELECTED

      7 GOAL 31 APPLY  RIGHT  CONDITION OPR-2 TO 14      (SUBGOAL OF 29)
                OPERATOR 16: (((~Q(U, Y) V... C) AND Q(U, Y)) YIELDS C)

      7 GOAL 32 SELECT FROM INITIAL-OBJ A/C LIT-SIGN OF  LEFT  CONDITION 16      (SUBGOAL OF 29)
                STATEMENT-3 SELECTED

      7 GOAL 33 APPLY  LEFT  CONDITION 16 TO STATEMENT-3      (SUBGOAL OF 29)
                OBJECT 17: (~P(F(U, Y), F(U, Y)) V... ~P(Y, F(U, Y)) V ~Q(F(U, Y), F(U, Y)))

  5 GOAL 34 TRANSFORM 17 INTO DESIRED-OBJ      (SUBGOAL OF 23)

    6 GOAL 35 REDUCE MAIN ON 17      (SUBGOAL OF 34)
```

Figure 63: (continued)

```
7 GOAL 36 APPLY OPR-1 TO 17      (SUBGOAL OF 35)

  8 GOAL 37 SELECT FROM  CONDITION OPR-1 A/C NO.-LIT OF 17      (SUBGOAL OF 36)
       NONE SELECTED

7 GOAL 38 APPLY OPR-2 TO 17      (SUBGOAL OF 35)

  8 GOAL 39 SELECT FROM  CONDITION OPR-2 A/C NO.-LIT OF 17      (SUBGOAL OF  LEFT  CONDITION 38)
       CONDITION  LEFT SELECTED

  8 GOAL 40 APPLY  LEFT  CONDITION OPR-2 TO 17      (SUBGOAL OF 38)
       OPERATOR 18:  (((-P(F(U, Y), F(U, Y)) V... -P(Y, F(U, Y)) V -Q(F(U, Y), F(U, Y
                     ))) AND P(F(U, Y), F(U, Y))) YIELDS (-P(Y, F(U, Y)) V -Q(F(U, Y)
                     , F(U, Y))))

  8 GOAL 41 SELECT FROM INITIAL-OBJ A/C LIT-SIGN OF  RIGHT  CONDITION 18      (SUBGOAL OF 38)
       STATEMENT-1 SELECTED

  8 GOAL 42 APPLY  RIGHT  CONDITION 18 TO STATEMENT-1      (SUBGOAL OF 38)
       OBJECT 19: (-P(Y, F(U, Y)) V... -Q(F(U, Y), F(U, Y)))

6 GOAL  43 TRANSFORM 19 INTO DESIRED-OBJ      (SUBGOAL OF 34)

  7 GOAL 44 REDUCE MAIN ON 19      (SUBGOAL OF 43)

    8 GOAL 45 APPLY OPR-1 TO 19      (SUBGOAL OF 44)

      9 GOAL 46 SELECT FROM  CONDITION OPR-1 A/C NO.-LIT OF 19      (SUBGOAL OF 45)
         NONE SELECTED

    8 GOAL 47 APPLY OPR-2 TO 19      (SUBGOAL OF 44)

      9 GOAL 48 SELECT FROM  CONDITION OPR-2 A/C NO.-LIT OF 19      (SUBGOAL OF  LEFT  CONDITION 47)
         CONDITION  LEFT SELECTED

      9 GOAL 49 APPLY  LEFT  CONDITION OPR-2 TO 19      (SUBGOAL OF 47)
         OPERATOR 20:  (((-P(Y, F(U, Y)) V... -Q(F(U, Y), F(U, Y))) AND P(Y, F(U, Y))) YIELDS
                       -Q(F(U, Y), F(U, Y)))

      9 GOAL 50 SELECT FROM INITIAL-OBJ A/C LIT-SIGN OF  RIGHT  CONDITION 20      (SUBGOAL OF 47)
         STATEMENT-1 SELECTED

      9 GOAL 51 APPLY  RIGHT  CONDITION 20 TO STATEMENT-1      (SUBGOAL OF 47)
         OBJECT 21: -Q(F(U, Y), F(U, Y))

  7 GOAL  52 TRANSFORM 21 INTO DESIRED-OBJ      (SUBGOAL OF 43)

    8 GOAL 53 REDUCE MAIN ON 21      (SUBGOAL OF 52)
```

Figure 63: (continued)

```
9 GOAL 54 APPLY OPR-1 TO 21      (SUBGOAL OF 53)

  10 GOAL 55 SELECT FROM  CONDITION OPR-1 A/C NO.-LIT OF 21      (SUBGOAL OF  LEFT  CONDITION 54)
    CONDITION  RIGHT SELECTED

  10 GOAL 56 APPLY  LEFT  CONDITION OPR-1 TO 21      (SUBGOAL OF 54)
    OPERATOR 22:  ((-Q(F(U, Y), F(U, Y)) AND Q(F(U, Y), F(U, Y))) YIELDS NULL)

  10 GOAL 57 SELECT FROM INITIAL-OBJ A/C LIT-SIGN OF  RIGHT  CONDITION 22      (SUBGOAL OF 54)
    14 SELECTED

  10 GOAL 58 APPLY  RIGHT  CONDITION 22 TO 14      (SUBGOAL OF 54)
    OBJECT 23: NULL

8  GOAL  59 TRANSFORM 23 INTO DESIRED-OBJ      (SUBGOAL OF 52)
SUCCESS
```

Figure 63: (continued)

OBJECT-SCHEMA is a single literal; otherwise, it must contain more than one literal. The purpose of these criteria will become clear in the discussion of the behavior of GPS on this task.

Like the selection criterion, the immediate operators of this task are given to GPS as IPL structures and do not appear in Fig. 61. The main function of the immediate operators is to permute the literals in a clause when necessary.

Behavior of GPS

Fig. 63 is the behavior of GPS in solving the task in Fig. 61. To achieve TOP-GOAL, GPS recognizes that INITIAL-OBJ is a SET of OBJECT-SCHEMAs and generates GOAL 2 to find the member of the set which appears to be most easily transformed into DESIRED-OBJ. STATEMENT-1 is selected because it contains the fewest number of literals, and GOAL 3 is created in an attempt to achieve GOAL 1. GPS notices that the value of the ATTRIBUTE, SYMBOL, at the TOP-NODE of the two OBJECT-SCHEMAs in GOAL 3 are different and creates GOAL 4 to reduce this difference. According to TABLE-OF-CONNECTIONS, OPR-1 has the capability of reducing the difference and thus GOAL 5 is created.

In attempting GOAL 5 GPS notices that OPR-1 has two OBJECT-SCHEMAs as an input and the TWO-INPUT-OPERATOR-METHOD is selected for achieving GOAL 5. The first step in this method is to decide which of the input forms (-B or B) corresponds to STATEMENT-1; GOAL 6 is created for this purpose. Since both input forms and STATEMENT-1 are clauses which contain only one literal, GPS decides that either input form can be used and selects the input form, -B. The next step in the TWO-INPUT-OPERATOR-METHOD matches the input form selected to STATEMENT-1 (GOAL 7) to determine if they can be made

identical through the substitution of variables. The result of GOAL 7 (OPERATOR 10) is OPR-1 after STATEMENT-1 has been matched to the input form, i.e., after P(U,Y) is substituted for -B. At this point, GPS knows that the input form of OPERATOR 10, P(U,Y) has already been supplied, even though Fig. 63 does not indicate this, and that the resultant object can be produced by supplying the other input. The other input can be any OBJECT-SCHEMA derived from INITIAL-OBJ; thus, the next step in the TWO-INPUT-OPERATOR-METHOD is GOAL 8, whose purpose is to select the OBJECT-SCHEMA to which OPERATOR 10 can most easily be applied. No OBJECT-SCHEMA in INITIAL-OBJ contains only a single literal whose SIGN is - and predicate letter is P and thus GOAL 8 as well as GOAL 5 fails.

In retrying GOAL 4, GPS generates GOAL 9 because OPR-2 may reduce the difference. GPS, by the result of GOAL 10, decides that STATEMENT-1 should be matched to the input form -B because it, like STATEMENT-1, contains only a single literal. GOAL 11 is successful and its result is the partially applied OPERATOR 11.

To find the other input for OPERATOR 11, GPS creates GOAL 12. STATEMENT-3 is selected because it contains more than one literal and one of them has - for its SIGN and P for its predicate letter. OPERATOR 11 is applied (GOAL 13) and produces OBJECT 12.

GPS tries to transform OBJECT 12 into the DESIRED-OBJ (GOAL 14). Since they are different, GOAL 15 is created, which GPS attempts to achieve by applying OPR-1 to OBJECT 12 (GOAL 16). GOAL 17 is created in order to decide which input form should be matched to OBJECT 12. Both input forms in OPR-1 contain only one literal and OBJECT 12 has two; hence GPS considers the application of OPR-1 infeasible. In attempting GOAL 15 again, GOAL 18

is created.

GPS continues in similar fashion until □ is finally produced as a result of GOAL 59 and the success of GOAL 60 indicates that the TOP-GOAL has been achieved.

Discussion

The formulation of this task for GPS is somewhat clumsy. To express the resolution principle, which is used as a single rule of inference by the predicate calculus theorem provers, three distinct operators are required. This clumsiness results from a lack of any notation for sets and any operations on sets in the representation of GPS; there is no convenient way to represent the union of two sets or the set formed by deleting a member from a set.

OBJECT-SCHEMAs are not represented by the disjunction of a set of literals. Rather disjunction is treated as a binary logical connective. Fig. 62.b illustrates that in this representation, the arguments of V are either literals or a disjunction of literals. Fig. 62.a is the representation of a clause comprised of a single literal. Such a clause is represented by the literal itself and thus contains no V.

The operators must distinguish between OBJECT-SCHEMAs which contain only a single literal and those which contain more than one literal. In the one case, the literal which is "deleted" is at the TOP-NODE and, in the other case, it is at the LEFT node. The three operators are required to take care of the three situations,

a. when both input clauses contain a single literal;
b. when only one input clause contains a single literal;
c. when neither input clause contains just one literal.

OPR-2 (used in situation b) can have the single literal either as its first input or its second input.

It is noteworthy that representing the resolution principle as three operators instead of one has a considerable effect on the problem solving. Often GPS tries the wrong operator first, e.g., GOAL 5, and does not realize the mistake until several GOALs have been generated. If the three operators were combined into one, the wrong operator could not possibly be selected. In solving the task in Fig. 61, 14 GOALs (a quarter of the GOALs) are generated in attempting to apply inapplicable operators.

On the other hand, the fact that there are three operators instead of one, is used beneficially by GPS. GPS attempts to apply OPR-1 and OPR-2 before applying OPR-3 because of their order in the TABLE-OF-CONNECTIONS. Since one of the inputs to both OPR-1 and OPR-2 contain only a single literal, the result of applying either has fewer literals than one of the inputs. Reducing the number of literals in a clause is desirable because the main objective of the task is to reduce the number of literals in a clause to zero; i.e., to produce □ . Applying OPR-1 and OPR-2 before applying OPR-3 is equivalent to the unit preference strategy used in Wos, et al [67].

The commutativity and associativity of V is implicitly taken into consideration in applying an operator. Whenever the match detects a V... versus V, an immediate operator juggles the order of the literals in an object until the SIGN and predicate letter of the first literal are the same in the object and input form of the operator.

For example, in GOAL 33

-Q(U,Y) V... C

is matched to Fig. 62.b. After detecting the V...versus V, an immediate

operator rewrites Fig. 62.b as Fig. 62.c and replaces the V... with a V. Since Fig. 62.c can be made to match the input form by substituting for variables, OBJECT 17 can be produced.

It is very important that the problem formulation takes advantage of the commutativity of the literals in a clause implicitly, instead of stating it explicitly, as an operator. Otherwise, the problem tree would be considerably larger and the problem solver would spend a large portion of its time commuting literals in a clause.

Perhaps the most instructive part of this example is the light it cast upon the evolution of problem solving programs. In LT, a theorem proving program for the prepositional calculus which is the predecessor of GPS, it was noted that the match routine was the source of most of the power of the program over a brute force search. (GPS may be considered as an attempt to generalize the match routine, based on that experience.) The first predicate calculus theorem prover did in fact use brute force search, Gilmore [17]. From an efficiency point of view the main effect of the resolution principle was to reintroduce the possibility of matching (gaining, thereby, a vast increase in power). And it is this feature that allows GPS to use the resolution principle in a natural way.

E. FATHER AND SONS TASK[9]

A father and his two sons want to cross a river. The only means of conveyance is a small boat whose capacity is 200 pounds. Each son weighs 100 pounds while the father weighs 200 pounds. Assuming that the father

[9]The source of this task is private communication from Paul Newell.

and either son can operate the boat, how can they all reach the other side of the river? Of course, there is no way to cross the river except by boat.

GPS Formulation

Fig. 64 gives a formulation of this task for GPS. TOP-GOAL is a statement of the problem. INITIAL-OBJ, whose tree structure is shown in Fig. 65, represents the situation in which the father, both of his sons, and the boat are on the LEFT bank of the river. The presence of the BOAT is designated the value, BOAT, of the ATTRIBUTE, BOAT; its absence is indicated by the absence of a value of the ATTRIBUTE, BOAT.

DESIRED-OBJ is the OBJECT-SCHEMA that represents the situation in which the father, his two sons, and the boat are on the RIGHT bank of the river.

In this task, the only operator is SAIL whose application has the effect of moving X fathers, Y sons, and the boat from the FROM-SIDE to the TO-SIDE. The first two TESTs in VAR-DOMAIN of SAIL indicate that FROM-SIDE and TO-SIDE stand for different banks of the river.

The third and fourth TESTS require that someone must be in the boat to operate it, and that the capacity of the boat must not be exceeded. WEIGHT is the EXPRES whose value is the number of hundred pounds in the boat. Thus, if the value of WEIGHT is greater than 0, someone must be in the boat and, if the value of WEIGHT is not greater than two, the capacity of the boat is not exceeded. Negative values for X and Y do not make sense.

The three TRANSFORMATIONs in SAIL move the BOAT, the FATHER, and the SONS, respectively, across the river. The four SETs, SIDES, 0-1, 0-1-2, and 1-2 are used in the specification of SAIL.

The six types of differences, F-L, F-R, S-L, S-R, D-L (dock at left),

```
DECLARE    (
   BOAT = ATTRIBUTE
   O-L = FEATURE
   O-R = FEATURE
   FATHERS = ATTRIBUTE
   FINAL-OBJ = OBJECT-SCHEMA
   FROM-SIDE = LOC-PROG
   F-L = FEATURE
   F-R = FEATURE
   INITIAL-OBJ = OBJECT-SCHEMA
   SAIL = MOVE-OPERATOR
   SONS = ATTRIBUTE
   SIDES = SET
   S-L = FEATURE
   S-R = FEATURE
   TO-SIDE = LOC-PROG
   WEIGHT = EXPRES
   0-1 = SET
   0-1-2 = SET
   1-2 = SET
         )
TASK-STRUCTURES    (
   TOP-GOAL = ( TRANSFORM THE INITIAL-OBJ INTO THE FINAL-OBJ . )
   INITIAL-OBJ = ( LEFT ( SONS 2 FATHERS 1 BOAT BOAT )
                   RIGHT ( SONS 0 FATHERS 0 ) )
   FINAL-OBJ = ( LEFT ( SONS 0 FATHERS 0 )
                 RIGHT ( FATHERS 1 SONS 2 BOAT BOAT ) )
```

Figure 64: The specification for GPS of the father and sons task.

```
WEIGHT = ( X + X + Y )
SAIL = $ SAIL THE BOAT FROM THE FROM-SIDE TO THE TO-SIDE WITH X FATHERS
         AND Y SONS IN IT. $
         ( CREATION-OPERATOR
           VAR-DOMAIN
           1. THE FROM-SIDE IS AN EXCLUSIVE-MEMBER OF THE SIDES .
           2. THE TO-SIDE IS AN EXCLUSIVE-MEMBER OF THE SIDES .
           3. Y IS A CONSTRAINED-MEMBER OF 0-1 . THE CONSTRAINT IS THAT
              THE WEIGHT IS IN-THE-SET 1-2 .
           4. X IS A CONSTRAINED-MEMBER OF 0-1-2 . THE CONSTRAINT IS THAT
              THE WEIGHT IS IN-THE-SET 1-2 .
           MOVES
           1. MOVE THE BOAT AT THE FROM-SIDE TO THE BOAT AT THE TO-SIDE .
           2. DECREASE BY THE AMOUNT X THE FATHERS AT THE FROM-SIDE AND
              ADD IT TO THE FATHERS AT THE TO-SIDE .
           3. DECREASE BY THE AMOUNT Y THE SONS AT THE FROM-SIDE AND ADD
              IT TO THE SONS AT THE TO-SIDE .   )
SIDES = ( LEFT RIGHT )
0-1 = ( 0 1 )
0-1-2 = ( 0 1 2 )
1-2 = ( 1 2 )
S-L = ( THE SONS AT THE LEFT )
S-R = ( THE SONS AT THE RIGHT )
F-L = ( THE FATHERS AT THE LEFT )
F-R = ( THE FATHERS AT THE RIGHT )
B-L = ( THE BOAT AT THE LEFT )
B-R = ( THE BOAT AT THE RIGHT )
```

Figure 64: (continued)

```
    BASIC-MATCH = ( COMP-FEAT-LIST ( F-R S-R D-R ) )
    COMP-OBJECTS = ( BASIC-MATCH )
    DIFF-ORDERING = ( 1. ( F-L F-R )
                      2. ( S-L S-R )
                      3. ( D-L D-R ) )
    TABLE-OF-CONNECTIONS = ( ( COMMON-DIFFERENCE SAIL ) )
    LIST-OF-VAR = ( FROM-SIDE TO-SIDE X Y )
    OBJ-ATTRIB = ( FATHERS SONS BOAT )

END
```

Figure 64: (continued)

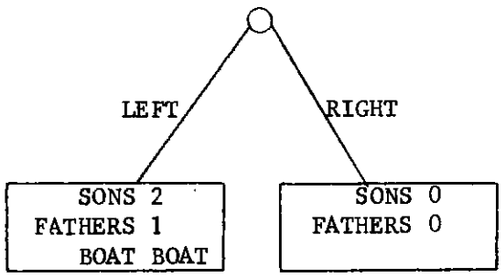


FIGURE 65. The tree structure representation of INITIAL-OBJ.

and D-R are all stated relative to the TOP-NODE of an OBJECT-SCHEMA. The types of differences which pertain to the FATHERs are most difficult according to DIFF-ORDERING, while the easiest types of differences are those which pertain to the BOAT.

BASIC-MATCH and COMPARE-OBJECTS indicate that the only types of differences detected by the match are those that pertain to the RIGHT bank of the river. These types of differences are sufficient because what is not at the RIGHT must be at the LEFT.

TABLE-OF-CONNECTIONS designates SAIL to be relevant to reducing all types of differences and thus gives GPS no selectivity. OBJ-ATTRIB is a list of the ATTRIBUTEs of this task and LIST-OF-VAR lists the variables which appear in Fig. 64.

NEW-OBJ is a selection criterion which was given to GPS as an IPL structure. It is used in several tasks and is described on page 171.

Behavior of GPS

Fig. 66 shows the behavior of GPS in solving the task in Fig. 1. In attempting TOP-GOAL, GPS notices that neither the father, his sons, nor the boat is at the RIGHT bank of the river. GOAL 2 is generated because, according to DIFF-ORDERING, F-R is more difficult than either D-R or S-R. To reduce this difference, SAIL, with X equal to 1, and TO-SIDE equal to LEFT, is applied to the INITIAL-OBJ (GOAL 3, OBJECT 5).

Since there are not enough SONS at the RIGHT in OBJECT 5, (GOAL 4, GOAL 5), GPS attempts to move two of them to the RIGHT (GOAL 6). GPS notices that before it can apply this operator, the BOAT, must be brought back to the LEFT (GOAL 7). Although bringing the BOAT back to the LEFT results in the INITIAL-OBJ (GOAL 8), it allows the two SONS to be moved to the RIGHT (GOAL 9, OBJECT 6).

```
1 TOP-GOAL TRANSFORM INITIAL-OBJ INTO FINAL-OBJ      (SUBGOAL OF NONE)

  2 GOAL 2 REDUCE F-R ON INITIAL-OBJ      (SUBGOAL OF TOP-GOAL)

    3 GOAL 3 APPLY SAIL WITH X = 1, TO-SIDE = RIGHT, TO INITIAL-OBJ      (SUBGOAL OF 2)
                SET: FROM-SIDE = LEFT, Y = 0
                OBJECT 5: (LEFT(FATHERS 0 SONS 2) RIGHT(FATHERS 1 SONS 0 BOAT BOAT))

  2 GOAL 4 TRANSFORM 5 INTO FINAL-OBJ      (SUBGOAL OF TOP-GOAL)

    3 GOAL 5 REDUCE S-R ON 5      (SUBGOAL OF 4)

      4 GOAL 6 APPLY SAIL WITH Y = 2, TO-SIDE = RIGHT, TO 5      (SUBGOAL OF 5)
                SET: FROM-SIDE = LEFT, X = 0

        5 GOAL 7 REDUCE D-L ON 5      (SUBGOAL OF 6)

          6 GOAL 8 APPLY SAIL WITH TO-SIDE = LEFT, TO 5      (SUBGOAL OF 7)
                SET: FROM-SIDE = RIGHT, X = 1, Y = 0
                OBJECT INITIAL-OBJ: (LEFT(FATHERS 1 SONS 2 BOAT BOAT) RIGHT(FATHERS 0 SONS 0))

        5 GOAL 9 APPLY SAIL WITH Y = 2, TO-SIDE = RIGHT, TO INITIAL-OBJ      (SUBGOAL OF 6)
                SET: FROM-SIDE = LEFT, X = 0
                OBJECT 6: (LEFT(FATHERS 1 SONS 0) RIGHT(FATHERS 0 SONS 2 BOAT BOAT))

  3 GOAL 10 TRANSFORM 6 INTO FINAL-OBJ      (SUBGOAL OF 4)

    4 GOAL 11 REDUCE F-R ON 6      (SUBGOAL OF 10)

      5 GOAL 12 APPLY SAIL WITH X = 1, TO-SIDE = RIGHT, TO 6      (SUBGOAL OF 11)
                SET: FROM-SIDE = LEFT, Y = 0

        6 GOAL 13 REDUCE D-L ON 6      (SUBGOAL OF 12)

          7 GOAL 14 APPLY SAIL WITH TO-SIDE = LEFT, TO 6      (SUBGOAL OF 13)
                SET: FROM-SIDE = RIGHT, X = 0, Y = 1
                OBJECT 7: (LEFT(FATHERS 1 SONS 1 BOAT BOAT) RIGHT(FATHERS 0 SONS 1))

        6 GOAL 15 APPLY SAIL WITH X = 1, TO-SIDE = RIGHT, TO 7      (SUBGOAL OF 26)
                SET: FROM-SIDE = LEFT, Y = 0
                OBJECT 8: (LEFT(FATHERS 0 SONS 1) RIGHT(FATHERS 1 SONS 1 BOAT BOAT))
```

Figure 66: The performance of GPS on the father and sons task.

```
4  GOAL 16 TRANSFORM 8 INTO FINAL-OBJ      (SUBGOAL OF 10)

  5 GOAL 17 REDUCE S-R ON 8      (SUBGOAL OF 16)

    6 GOAL 18 APPLY SAIL WITH Y = 1, TO-SIDE = RIGHT, TO 8      (SUBGOAL OF 17)
            SET: FROM-SIDE = LEFT, X = 0

      7 GOAL 19 REDUCE D-L ON 8      (SUBGOAL OF 18)

        8 GOAL 20 APPLY SAIL WITH TO-SIDE = LEFT, TO 8      (SUBGOAL OF 19)
              SET: FROM-SIDE = RIGHT, Y = 1, X = 0
              OBJECT 9: (LEFT(FATHERS 0 SONS 2 BOAT BOAT) RIGHT(FATHERS 1 SONS 0))

      7 GOAL 21 APPLY SAIL WITH Y = 1, TO-SIDE = RIGHT, TO 9      (SUBGOAL OF 18)
            SET: FROM-SIDE = LEFT, X = 0
            OBJECT 8: (LEFT(FATHERS 0 SONS 1) RIGHT(FATHERS 1 SONS 1 BOAT BOAT))
4  GOAL 16 TRANSFORM 8 INTO FINAL-OBJ      (SUBGOAL OF 10)

  5  GOAL 23 TRANSFORM 4 INTO FINAL-OBJ      (SUBGOAL OF TOP-GOAL)

    6 GOAL 24 SELECT FROM 4 A/C NEW-OBJ OF FINAL-OBJ      (SUBGOAL OF 23)
            9 SELECTED

    6  GOAL 25 TRANSFORM 7 INTO FINAL-OBJ      (SUBGOAL OF 23)

      7 GOAL 26 REDUCE F-R ON 7      (SUBGOAL OF 25)

    6 GOAL 15 APPLY SAIL WITH X = 1, TO-SIDE = RIGHT, TO 7      (SUBGOAL OF 26)
            SET: FROM-SIDE = LEFT, Y = 0
            OBJECT 8: (LEFT(FATHERS 0 SONS 1) RIGHT(FATHERS 1 SONS 1 BOAT BOAT))
4  GOAL 16 TRANSFORM 8 INTO FINAL-OBJ      (SUBGOAL OF 10)

4 GOAL 11 REDUCE F-R ON 6      (SUBGOAL OF 10)

  5  GOAL 23 TRANSFORM 4 INTO FINAL-OBJ      (SUBGOAL OF TOP-GOAL)

    6 GOAL 24 SELECT FROM 4 A/C NEW-OBJ OF FINAL-OBJ      (SUBGOAL OF 23)
            9 SELECTED

    6  GOAL 30 TRANSFORM 9 INTO FINAL-OBJ      (SUBGOAL OF 23)
```

Figure 66: (continued)

```
7 GOAL 31 REDUCE S-R ON 9       (SUBGOAL OF 30)

  8 GOAL 32 APPLY SAIL WITH Y = 2, TO-SIDE = RIGHT, TO 9       (SUBGOAL OF 31)
        SET: FROM-SIDE = LEFT, X = 0
        OBJECT 12: (LEFT(FATHERS 0 SONS 0) RIGHT(FATHERS 1 SONS 2 BOAT BOAT))

7  GOAL 33 TRANSFORM 12 INTO FINAL-OBJ       (SUBGOAL OF 30)
SUCCESS
```

Figure 66: (continued)

After one of the SONS brings the BOAT to LEFT, GPS moves the father to the RIGHT (GOAL 15). GPS attempts to move the remaining son at the LEFT across the river by bringing the BOAT back to the LEFT (GOAL 20). Since GPS does not realize that the other son was brought to the LEFT with the BOAT, only one son is moved to the RIGHT (GOAL 21) which causes GOAL 16 to be regenerated.

At this point, GPS realizes that it is in trouble and looks for something new to do. GOAL 25 is created beçause OBJECT 7 has never been transformed into the FINAL-OBJ and because it is derived from INITIAL-OBJ. (OBJECT 4 in GOAL 23 is the SET of all objects derived from INITIAL-OBJ. It is generated internally by GPS.)

Since GOAL 25 does not lead to any new results, GOAL 30 is generated in an attempt to generate a new GOAL. After moving two sons across the river in OBJECT 9, GPS notices that it is successful (GOAL 33).

Discussion

This task is very similar to the missionaries and cannibals task. Both tasks involve moving two different kinds of people across a river in a small boat. But their formulations for GPS are quite different, in that none of the operators, objects, or differences are the same. The father and sons task cannot be given to GPS in terms of the missionaries and cannibals task, e.g., the father and sons task is the same as the missionaries and cannibals task except that nobody can be eaten and there is only one father, etc. On the other hand, after giving the task of integrating an expression to GPS, only TOP-GOAL had to be changed in order for GPS to integrate a different expression.

F. MONKEY TASK

This task was invented by McCarthy [27] as a typical problem for the Advice Taker program, McCarthy [26]. In a room is a monkey, a box, and some bananas hanging from the ceiling. The monkey wants to eat the bananas, but he cannot reach them unless he is standing on the box when it is sitting under the bananas. How can the monkey get the bananas? The answer is that the monkey must move the box under the bananas and climb on the box before he can reach the bananas. The problem originates in the study of the problem solving ability of primates. Its interest lies not in its difficulty, but in its being an example of a problem subject to common sense reasoning.

GPS Formulation

In the formulation of this task shown in Fig. 67, INITIAL-OBJ represents the monkey in the room. INITIAL-OBJ is an OBJECT-SCHEMA consisting of only a single node with the three ATTRIBUTEs listed in OBJ-ATTRIB. There are four operators which represent the various acts that the monkey can perform. CLIMB is the operator whose application corresponds to the monkey climbing on the box. An application of WALK corresponds to the monkey walking to the place, X, which must on on the floor of the room; i.e., in the set PLACES. MOVE-BOX represents the fact that the monkey can move the box to any place on the floor of the room and GET-BANANAS is the statement that the monkey can get the bananas provided that the box is under the bananas and that he is standing on the box.

The three types of differences, D1, D2, D3, of this task correspond to an incorrect value of one of the three ATTRIBUTEs (OBJ-ATTRIB) of this task: D1 to MONKEY'S-PLACE, D2 to BOX'S-PLACE, and D3 to CONTENTS-OF-MONKEY'S-

```
DECLARE   (
   BOX'S-PLACE = ATTRIBUTE
   CLIMB     = MOVE-OPERATOR
   CONTENTS-OF-MONKEY'S-HAND = ATTRIBUTE
   DESIRED-OBJ = DESCRIBED-OBJ
   D1 = FEATURE
   D2 = FEATURE
   D3 = FEATURE
   GET-BANANAS = MOVE-OPERATOR
   INITIAL-OBJ = OBJECT-SCHEMA
   MONKEY'S-PLACE = ATTRIBUTE
   MOVE-BOX = MOVE-OPERATOR
   PLACES = SET
   WALK      = MOVE-OPERATOR
   )
TASK-STRUCTURES   (
   TOP-GOAL = ( TRANSFORM THE INITIAL-OBJ INTO THE DESIRED-OBJ . )
   INITIAL-OBJ = ( MONKEY'S-PLACE PLACE-1 BOX'S-PLACE PLACE-2
                   CONTENTS-OF-MONKEY'S-HAND EMPTY )
   OBJ-ATTRIB = ( MONKEY'S-PLACE BOX'S-PLACE CONTENTS-OF-MONKEY'S-HAND )
   DESIRED-OBJ = ( TEX-DESCRIPTION
                   THE CONTENTS-OF-MONKEY'S-HAND EQUALS BANANAS . )
   PLACES = ( PLACE-1 PLACE-2 UNDER-BANANAS . )
   CLIMB = ( CREATION-OPERATOR
             PRETESTS
             1. THE MONKEY'S-PLACE EQUALS THE BOX'S-PLACE .
             MOVES
```

Figure 67: The specification for GPS of the monkey task.

```
          1. COPY ON-BOX AT THE MONKEY'S-PLACE . )
WALK = ( CREATION-OPERATOR
         VAR-DOMAIN
         X IS IN-THE-SET OF PLACES .
         MOVES
         COPY X AT THE MONKEY'S-PLACE . )
MOVE-BOX = ( CREATION-OPERATOR
             VAR-DOMAIN
             1. X IS IN-THE-SET OF PLACES .
             PRETESTS
             1. THE MONKEY'S-PLACE IS IN-THE-SET OF PLACES .
             2. THE MONKEY'S-PLACE EQUALS THE BOX'S-PLACE .
             MOVES
             1. COPY X AT THE MONKEY'S-PLACE .
             2. COPY X AT THE BOX'S-PLACE . )
GET-BANANAS = ( CREATION-OPERATOR
                PRETESTS
                1. THE BOX'S-PLACE EQUALS UNDER-BANANAS .
                2. THE MONKEY'S-PLACE EQUALS ON-BOX .
                MOVES
                1. COPY BANANAS AT THE CONTENTS-OF-MONKEY'S-HAND . )
D1 = ( MONKEY'S-PLACE )
D2 = ( BOX'S-PLACE )
D3 = ( CONTENTS-OF-MONKEY'S-HAND )
DIFF-ORDERING = ( D3 D2 D1 )
TABLE-OF-CONNECTIONS = ( ( COMMON-DIFFERENCE WALK CLIMB MOVE-BOX
                                             GET-BANANAS ) )
```

Figure 67: (continued)

```
    LIST-OF-VAR * ( X )
    )
END
```

Figure 67: (continued)

HAND. DIFF-ORDERING indicates that it is most difficult to change the contents of the monkey's hand while the monkey can easily change his position. TABLE-OF-CONNECTIONS contains no information about the desirability of operators, except that operators are desirable to reducing differences. LIST-OF-VAR indicates that X is the only variable in Fig. 67.

Behavior of GPS

Fig. 68 shows how GPS solved the problem in Fig. 67. The monkey cannot reach the bananas in initial configuration of the room (GOAL 1, GOAL 2, GOAL 3) because the box is not under the bananas (and because the monkey is not on the box, which is less important). In order to move the box under the bananas (GOAL 4, GOAL 5), the monkey must be standing beside the box; consequently, the monkey walks to the box (GOAL 6, GOAL 7) which results in the new configuration of the room--OBJECT 4. After the monkey moves the box under the bananas (GOAL 8, OBJECT 5), he still cannot reach the bananas (GOAL 9) because he is not standing on the box. So, the monkey climbs on the box (GOAL 10, GOAL 11, OBJECT 6) and finally plucks the bananas from the ceiling (GOAL 11, GOAL 12, OBJECT 7).

Discussion

INITIAL-OBJ and all OBJECT-SCHEMAs derived from INITIAL-OBJ are models of the various configurations of the room. They are not complete models of the room, e.g., none of them contain any information about things hanging from the ceiling. But they contain sufficient detail of the room for this simple problem.

The representation of the problem in an Advice Taker problem solver; e.g., the program developed by Black [4] is quite different. There, the initial configuration of the room is not a single entity that models the situation, as in GPS's representation, but a group of independent linguistic expressions

```
1 TOP-GOAL TRANSFORM INITIAL-OBJ INTO DESIRED-OBJ      (SUBGOAL OF NONE)

  2 GOAL 2 REDUCE D3 ON INITIAL-OBJ      (SUBGOAL OF TOP-GOAL)

    3 GOAL 3 APPLY GET-BANANAS TO INITIAL-OBJ      (SUBGOAL OF 2)

      4 GOAL 4 REDUCE D2 ON INITIAL-OBJ      (SUBGOAL OF 3)

        5 GOAL 5 APPLY MOVE-BOX WITH X = UNDER-BANANAS, TO INITIAL-OBJ      (SUBGOAL OF 4)

          6 GOAL 6 REDUCE D1 ON INITIAL-OBJ      (SUBGOAL OF 5)

            7 GOAL 7 APPLY WALK WITH X = PLACE-2, TO INITIAL-OBJ      (SUBGOAL OF 6)
                    OBJECT 4: (MONKEY'S-PLACE PLACE-2 BOX'S-PLACE PLACE-2 CONTENTS-OF-MONKEY'S-HAND
                    EMPTY)

            8 GOAL 8 APPLY MOVE-BOX WITH X = UNDER-BANANAS, TO 4      (SUBGOAL OF 5)
                    OBJECT 5: (MONKEY'S-PLACE UNDER-BANANAS BOX'S-PLACE UNDER-BANANAS CONTENTS-OF-MONKEY'S-HAND
                    EMPTY)

        4 GOAL 9 APPLY GET-BANANAS TO 5      (SUBGOAL OF 3)

          5 GOAL 10 REDUCE D1 ON 5      (SUBGOAL OF 9)

            6 GOAL 11 APPLY CLIMB TO 5      (SUBGOAL OF 10)
                    OBJECT 6: (MONKEY'S-PLACE ON-BOX BOX'S-PLACE UNDER-BANANAS CONTENTS-OF-MONKEY'S-HAND
                    EMPTY)

            5 GOAL 12 APPLY GET-BANANAS TO 6      (SUBGOAL OF 9)
                    OBJECT 7: (MONKEY'S-PLACE ON-BOX BOX'S-PLACE UNDER-BANANAS CONTENTS-OF-MONKEY'S-HAND
                    BANANAS)

    2  GOAL 13 TRANSFORM 7 INTO DESIRED-OBJ      (SUBGOAL OF TOP-GOAL)

SUCCESS
```

Figure 68: The performance of GPS on the monkey task.

that describe the situation. In solving this task, the program deduces new linguistic expressions and not new room configurations. For example, in solving this task, Black's program deduces the linguistic expressions,

a. The monkey can move the box under the bananas.
b. The monkey can stand on the box when it is under the bananas.

These linguistic expressions only describe part of a possible room configuration in much the same way the DESIRED-OBJ only describes part of a room configuration.

Both representations have advantages. Linguistic expressions are good for representing imperfect information which is difficult to represent in a model, e.g.,

The monkey is in one of two places.

On the other hand, models contain implicit information which need not be stated explicitly; e.g.,

the monkey can only be in one place at a time,

or

two squares are adjacent on a chess board.

Which of these two representations is better probably depends upon the task and for some tasks, such as the monkey task, they are probably equally good. This issue is discussed in Newell and Ernst [16].

G. THREE COINS PUZZLE

In this task, Filipiak [13], there are three coins setting on a table. Both the first and third coins show tails, while the second coin shows heads. The problem is to make all three coins the same--either heads or tails--in precisely three moves. Each move consists of "turning over" any two of the three coins. For example, if the first move consisted of turning over the first and third coins, all of the coins would be heads in the resulting situation. But the task is not solved because only one move was taken instead of the required three.

GPS Formulation

In Fig. 69 which gives the GPS formulation of the task, INITIAL-OBJ represents the situation in which the first and third coins are tails and the second coin is heads. The tree structure representation of INITIAL-OBJ is given in Fig. 70. Each node, except the TOP-NODE, represents a coin and has two ATTRIBUTEs, BOTTOM and TOP. The values of these ATTRIBUTEs are the side of the coin facing the table and the side of the coin showing, respectively. The ATTRIBUTEs of the TOP-NODE--MOVES-TAKEN and MOVES-REMAINING--keep track of the number of moves which are involved in producing the OBJECT-SCHEMA.

The DESIRED-OBJ represents the OBJECT-SCHEMAs in which the TOP of all three coins are the same and the MOVES-REMAINING is 0 (precisely three moves were involved in producing the OBJECT-SCHEMA). TOP-GOAL is the statement of the problem.

The only operator in this task is FLIP-COINS which "turns over" any two of the three coins. FLIP-COINS also increments by 1 the value of the MOVES-TAKEN and decrements by 1 the value of the MOVES-REMAINING.

```
RENAME   (
   COIN-1 = FIRST
   COIN-2 = SECOND
   COIN-3 = THIRD
     )
DECLARE   (
   BOTTOM = ATTRIBUTE
   COINS = SET
   DESIRED-OBJ = DESCRIBED-OBJ
   D1 = FEATURE
   D2 = FEATURE
   D3 = FEATURE
   D4 = FEATURE
   FLIP-COINS = MOVE-OPERATOR
   INITIAL-OBJ * OBJECT-SCHEMA
   MOVES-REMAINING = ATTRIBUTE
   MOVES-TAKEN = ATTRIBUTE
   TOP = ATTRIBUTE
   X = LOG-PROG
   Y = LOG-PROG
   )
TASK-STRUCTURES  (
   TOP-GOAL = ( TRANSFORM THE INITIAL-OBJ INTO THE DESIRED-OBJ . )
   INITIAL-OBJ = ( MOVES-REMAINING 3 MOVES-TAKEN 0
                   COIN-1 ( BOTTOM HEADS TOP TAILS )
                   COIN-2 ( BOTTOM TAILS TOP HEADS )
                   COIN-3 ( BOTTOM HEADS TOP TAILS ) )
```

Figure 69: The specification for GPS of the three coin puzzle.

```
DESIRED-OBJ = ( TEX-DESCRIPTION
                1. THE TOP OF COIN-2 EQUALS THE TOP OF COIN-3 .
                2. THE TOP OF COIN-3 EQUALS THE TOP OF COIN-1 .
                3. THE MOVES-REMAINING EQUALS 0 . )
COINS = ( COIN-1 COIN-2 COIN-3 )
FLIP-COINS = ( $ TURN COINS X AND Y OVER $ CREATION-OPERATOR
               VAR-DOMAIN
               1. X IS AN EXCLUSIVE-MEMBER OF THE COINS .
               2. Y IS AN EXCLUSIVE-MEMBER OF THE COINS .
               PRETESTS
               1. THE MOVES-REMAINING IS GREATER-THAN 0 .
               MOVES
               1. DECREASE BY THE AMOUNT 1 THE MOVES-REMAINING AND
                  ADD IT TO THE MOVES-TAKEN .
               2. MOVE THE BOTTOM OF X TO THE TOP OF X .
               3. MOVE THE BOTTOM OF Y TO THE TOP OF Y .
               4. MOVE THE TOP OF X TO THE BOTTOM OF X .
               5. MOVE THE TOP OF Y TO THE BOTTOM OF Y .
D1 = ( TOP OF COIN-1 )
D2 = ( TOP OF COIN-2 )
D3 = ( TOP OF COIN-3 )
D4 = ( MOVES-REMAINING )
DIFF-ORDERING = ( ( D1 D2 D3 D4 ) )
TABLE-OF-CONNECTIONS = ( ( COMMON-DIFFERENCE FLIP-COINS ) )
LIST-OF-VAR = ( X Y )
OBJ-ATTRIB = ( TOP BOTTOM MOVES-REMAINING MOVES-TAKEN )
)
```

Figure 69: (continued)

```
END
```

Figure 69: (continued)

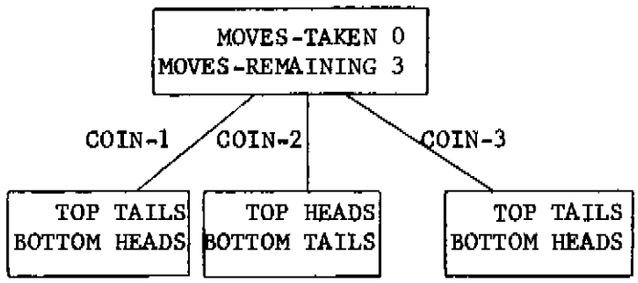


FIGURE 70. The tree structure representation of INITIAL-OBJ in the three coins puzzle.

The types of differences of this task are D1, D2, D3, (the TOPs of the various coins) and D4 (MOVE-REMAINING). Others, such as BOTTOM-OF-COIN-1 were not included because they could never be detected. DIFF-ORDERING signifies that all of the types of differences are equally difficult to reduce. TABLE-OF-CONNECTIONS indicates that FLIP-COINS is desirable to reducing any type of difference.

LIST-OF-VAR designates X and Y to be variables and the ATTRIBUTEs of this task are listed in OBJ-ATTRIB.

Behavior of GPS

Fig. 71 is the behavior of GPS in finding a solution to the task in Fig. 69. GPS notices that COIN-2 is not the same as COIN-3 (TOP-GOAL) and, in an attempt to reduce this difference (GOAL 2), GOAL 3 is created. OBJECT 4 is produced by "turning over" the first and second coins.

Since in OBJECT 4, COIN-3 is not the same as COIN-1 (GOAL 4, GOAL 5), GPS turns over the first and third coins (GOAL 6) to produce OBJECT 5. Again, COIN-2 is not the same as COIN-3 and the first and second coins are turned over (GOAL 7, GOAL 8, GOAL 9). In attempting GOAL 10, GPS notices that it has solved the problem.

Discussion

In the DESIRED-OBJ, the order of the arguments (for expedience) is such that GPS does not make a mistake. GPS would make a mistake if the TESTs in the DESIRED-OBJ were

1. THE TOP OF COIN-1 EQUALS THE TOP OF COIN-2.
2. THE TOP OF COIN-2 EQUALS THE TOP OF COIN-3.

Using this DESIRED-OBJ, the TOPs of the coins would not all be the same in the result of the ninth GOAL. FLIP-COINS could not be applied to this object

```
1 TOP-GOAL TRANSFORM INITIAL-OBJ INTO DESIRED-OBJ      (SUBGOAL OF NONE)

  2 GOAL: 2 REDUCE D2 ON INITIAL-OBJ      (SUBGOAL OF TOP-GOAL)

    3 GOAL 3 APPLY FLIP-COINS WITH X = COIN-2, TO INITIAL-OBJ      (SUBGOAL OF 2)
                SET:  Y = COIN-1
                OBJECT 4: (MOVES-REMAINING 2 MOVES-TAKEN 1 COIN-1(TOP HEADS BOTTOM TAILS) COIN-2
                          (TOP TAILS BOTTOM HEADS) COIN-3(TOP TAILS BOTTOM HEADS))

  2  GOAL 4 TRANSFORM 4 INTO DESIRED-OBJ      (SUBGOAL OF TOP-GOAL)

    3 GOAL 5 REDUCE D3 ON 4      (SUBGOAL OF 4)

      4 GOAL 6 APPLY FLIP-COINS WITH X = COIN-3, TO 4      (SUBGOAL OF 5)
                SET:  Y = COIN-1
                OBJECT 5: (MOVES-REMAINING 1 MOVES-TAKEN 2 COIN-1(TOP TAILS BOTTOM HEADS) COIN-2
                          (TOP TAILS BOTTOM HEADS) COIN-3(TOP HEADS BOTTOM TAILS))

    3  GOAL 7 TRANSFORM 5 INTO DESIRED-OBJ      (SUBGOAL OF 4)

      4 GOAL 8 REDUCE D2 ON 5      (SUBGOAL OF 7)

        5 GOAL 9 APPLY FLIP-COINS WITH X = COIN-2, TO 5      (SUBGOAL OF 8)
                SET:  Y = COIN-1
                OBJECT 6: (MOVES-REMAINING 0 MOVES-TAKEN 3 COIN-1(TOP HEADS BOTTOM TAILS) COIN-2
                          (TOP HEADS BOTTOM TAILS) COIN-3(TOP HEADS BOTTOM TAILS))

      4  GOAL 10 TRANSFORM 6 INTO DESIRED-OBJ      (SUBGOAL OF 7)

SUCCESS
```

Figure 71: The performance of GPS on the three coin puzzle.

because the MOVES-REMAINING is 0. (DECREASE requires its first argument to be greater than 0.) Consequently, TRY-OLD-GOALS-METHOD would be evoked which would cause a different feasible variable specification in applying an operator. This new result of an old APPLY-GOAL, shortly would lead to a solution.

The two ATTRIBUTEs, TOP and BOTTOM, could be replaced by a single ATTRIBUTE, ORIENTATION, whose value would be either HEADS or TAILS. In this formulation, the second and third TRANSFORMATIONs of FLIP-COINS would be replaced by the single TRANSFORMATION,

MOVE-FUNCTION OF THE ORIENTATION OF X TO THE
ORIENTATION OF X, THE FUNCTION IS F1.

F1 is the function whose value is TAILS when its argument is HEADS and vice-versa.

The most interesting aspect of this task is that its solution is constrained to a fixed number of operator applications. Many other tasks have this same property[10]. This property can be represented by associating with each object a counter which indicates the number of operator applications involved in producing the object.

H. PARSING SENTENCES

Generative grammars of certain languages can be defined by a set of phase structure rules, Chomsky [6]. Words in the language are divided into classes called parts of speech. If a word can be used as the part of speech, α, it is a member of the class, α. In general, a word is a member of

[10]The match-stick problems used in Katona [20] is another kind of task which has this property.

several parts of speech classes. In Fig. 72, the part of speech class α is indicated by <α>. The meaning, for example, of rule 3 is that an adjective phrase (AP), followed by a word that can be used as the part of speech, adjective, is an adjective phrase according to the grammar.

Parsing sentences can be accomplished by using these rules as replacement rules, i.e., any occurrence of the right side of a rule can be replaced by the left side of a rule. Consider the example of parsing, according to the grammar in Fig. 72, the sentence,

Free variables cause confusion.

Assuming that 'free' can be used as an adjective, an application of rule 4 produces

NP cause confusion.

The use of rules 6 and 8 (assuming that 'cause' is a verb and 'confusion' is a noun) yields

NP VP NP.

which is a sentence according to rule 1.

GPS Formulation

Fig. 73 is the GPS formulation of the task of parsing, according to the grammar in Fig. 72, the sentence,

Free variables cause confusion.

INITIAL-OBJ represents the sentence to be parsed. After translation, it is converted by a special routine for this task to the tree structure representation in Fig. 74.

The DESIRED-OBJ, which is also processed by the conversion routine,

1. S ← NP VP NP .
2. S ← NP VBP AP .
3. AP ← AP <adjective>
4. AP ← <adjective>
5. NP ← AP <noun>
6. NP ← <noun>
7. VP ← <adverb> <verb>
8. VP ← <verb>
9. VBP ← <adverb> <verb-be>
10. VBP ← <verb-be>

Definition of Symbols in the Above:

S---sentence
NP--noun phrase
AP--adjective phrase
VP--verb phrase

FIGURE 72. Phase structure rules for a simplified form of English.

```
RENAME   (
   NEXT = FIRST
   NEXT-OF-NEXT = FIRST-FIRST
     )
DECLARE  (
   FREE = SET
   VARIABLES = SET
   CAUSE = SET
   CONFUSION = SET
   PS = ATTRIBUTE
   WORD = ATTRIBUTE
   D1 = FEATURE
   A1 = MOVE-OPERATOR
   A2 = MOVE-OPERATOR
   N = MOVE-OPERATOR
   N1 = MOVE-OPERATOR
   V1 = MOVE-OPERATOR
   V2 = MOVE-OPERATOR
   V-B1 = MOVE-OPERATOR
   V-B2 = MOVE-OPERATOR
     )
LIST (
   DESIRED-OBJ = ( SENTENCE )
   INITIAL-OBJ = ( FREE VARIABLES CAUSE CONFUSION PERIOD )
   S1 = ( ( NOUN-PHRASE VERB-PHRASE NOUN-PHRASE PERIOD ) YIELDS SENTENCE )
   S2 = ( ( NOUN-PHRASE VERB-BE-PHRASE ADJECTIVE-PHRASE PERIOD ) YIELDS
          SENTENCE )
```

Figure 73: The specification for GPS of the task of parsing a sentence.

```
    )
TASK-STRUCTURES  (
   TOP-GOAL = ( TRANSFORM THE INITIAL-OBJ INTO THE DESIRED-OBJ . )
   OBJ-ATTRIB = ( PS WORD )
   FREE = ( NOUN ADJECTIVE VERB ).
   VARIABLES = ( NOUN )
   CAUSE = ( NOUN VERB )
   CONFUSION = ( NOUN )
   N = (   $ ADJ-PHRASE NOUN $ CREATION-OPERATOR
           PRETESTS
           1. NOUN IS IN-THE-SET OF THE NEXT WORD .
           2. ADJECTIVE-PHRASE EQUALS THE PS .
           MOVES
           1. COPY NOUN-PHRASE AT THE PS .
           2. MOVE THE NEXT-OF-NEXT TO THE NEXT . )
   A1 = (  $ADJ-PHRASE ADJ$  CREATION-OPERATOR
           PRETESTS
           1. ADJECTIVE IS IN-THE-SET OF THE NEXT WORD .
           2. THE PS EQUALS ADJECTIVE-PHRASE .
           MOVES
           1. MOVE THE NEXT-OF-NEXT TO THE NEXT .
           2. COPY ADJECTIVE-PHRASE AT THE PS .      )
   A2 = (  $ADJ$  CREATION-OPERATOR
           PRETESTS
           1. ADJECTIVE IS IN-THE-SET OF THE WORD .
           MOVES
           1. REMOVE THE WORD .
```

Figure 73: (continued)

```
        2. COPY ADJECTIVE-PHRASE AT THE PS .      )
N1 = ( $ NOUN $ CREATION-OPERATOR
     PRETESTS
     1. NOUN IS IN-THE-SET OF THE WORD .
     MOVES
     2. REMOVE THE WORD .
     3. COPY NOUN-PHRASE AT THE PS . )
V1 = (  $ADVERB VERB$  CREATION-OPERATOR
        PRETESTS
        1. VERB IS IN-THE-SET OF THE NEXT WORD .
        2. ADVERB IS IN-THE-SET OF THE WORD .
        MOVES
        1. REMOVE THE WORD .
        2. COPY VERB-PHRASE AT THE PS .
        3. MOVE THE NEXT-OF-NEXT TO THE NEXT . )
V2 = (  $VERB$ CREATION-OPERATOR
        PRETESTS
        1. VERB IS IN-THE-SET OF THE WORD .
        MOVES
        1. REMOVE THE WORD  .
        2. COPY VERB-PHRASE AT THE PS .      )
V-B1 = ( $ADVERB VERB-BE$  CREATION-OPERATOR
        PRETESTS
        1. VERB-BE IS IN-THE-SET OF THE NEXT WORD .
        2. ADVERB IS IN-THE-SET OF THE WORD .
        MOVES
        1. REMOVE THE WORD .
        2. COPY VERB-BE-PHRASE AT THE PS .
```

Figure 73: (continued)

```
        3. MOVE THE NEXT-OF-NEXT TO THE NEXT . )
   V-B2 = ( $VERB-BE$  CREATION-OPERATOR
        PRETESTS
        1. VERB-BE IS IN-THE-SET OF THE WORD .
        MOVES
        1. REMOVE THE WORD .
        2. COPY VERB-BE-PHRASE AT THE PS .    )
   D1 = ( PS )
   COMPARE-OBJECTS = ( BASIC-MATCH )
   BASIC-MATCH = ( COMP-FEAT-LIST ( D1 ) SUBEXPRESSIONS )
   DIFF-ORDERING = ( D1 )
   TABLE-OF-CONNECTIONS = ( ( D1 S1 S2 V1 V2 V-B1 V-B2 A1 A2 N N1 ) )
   LIST-OF-OPR = ( S1 S2 )
   )
END
```

Figure 73: (continued)

is an OBJECT-SCHEMA consisting of a single node which has SENTENCE as the value of the ATTRIBUTE, PS. PS stands for part of speech (which is a poor name for the ATTRIBUTE; grammatical type would be better). TOP-GOAL is the statement that the problem is to show that INITIAL-OBJ is a SENTENCE.

There are only two ATTRIBUTEs (OBJ-ATTRIB) in this formulation: The value of WORD is always an ENGLISH word and the value of PS is always a grammatical type defined by one of the rules of the grammar, i.e., the correspondent to the left side of one of the ten rules in Fig. 72. Each node of an OBJECT-SCHEMA represents either a part of speech (PS) or an English word. Thus, precisely one of the two ATTRIBUTEs, WORD and PS, has a value at every node of every OBJECT-SCHEMA. This convention is implicit in the formulation of the operators of this task.

There are ten operators, one corresponding to each of the ten rules in Fig. 72. The function of each of these operators is to replace an occurrence of the left side of the rule with the right side of the rule. S1 in Fig. 73 corresponds to the first rule in Fig. 72. It is a FORM-OPERATOR, which is converted (by the same special routine which converts INITIAL-OBJ, DESIRED-OBJ, and S2) to the tree structure in Fig. 75. Similarly, S2 corresponds to the second rule in Fig. 72.

Each WORD in INITIAL-OBJ is the SET of the parts of speech for which the WORD can be used. (Since PERIOD cannot be used as any part of speech, it is not a SET but a CONSTANT.) For example, CAUSE is the SET of two elements, NOUN and VERB, because CAUSE can be used as either a noun or a verb. The first TEST in the PRETESTS of N, which is the MOVE-OPERATOR representation of rule 6 in Fig. 72, is satisfied if the NEXT WORD can be used as a NOUN; thus, the TEST is satisfied if the NEXT WORD is CAUSE.

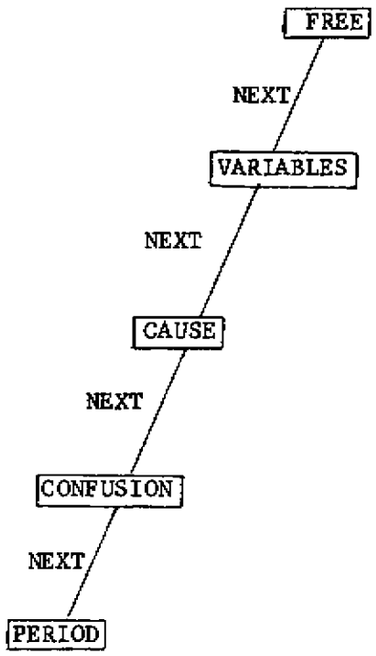
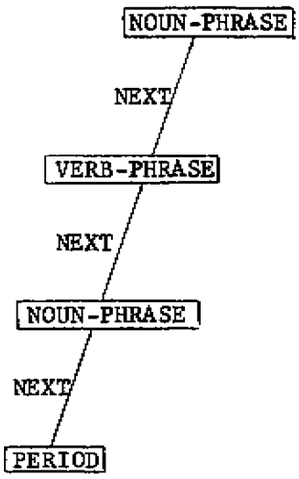


FIGURE 74. The tree structure representation of INITIAL-OBJ. The words at the nodes are values of the ATTRIBUTE, WORD.

Input form:



Output form: SENTENCE

FIGURE 75. The tree structure representation of the operator, S1. The words at the nodes are values of the ATTRIBUTE, PS except for PERIOD which is a value of WORD.

The first TRANSFORMATION in N changes the PS to be NOUN PHRASE. (According to the second TEST in PRETESTS, PS has the value, ADJECTIVE-PHRASE.) The second TRANSFORMATION of N has the effect of deleting the NEXT WORD from the string of WORDs and PSs represented by the OBJECT-SCHEMA.

The only type of difference used in this task is D1 which pertains to the value of the ATTRIBUTE, PS. The type of difference that refers to the value of WORD, is not used because the value of this ATTRIBUTE cannot be changed. (Some of the operators REMOVE the value of WORD from a node but none replace it with a different value.)

COMPARE-OBJECTS and BASIC-MATCH indicate that the match tests if the values of PS at all corresponding nodes of two OBJECT-SCHEMAs are identical. DIFF-ORDERING is a formality for this task because D1 is the only type of difference, and TABLE-OF-CONNECTIONS signifies that all of the operators are relevant to reducing D1. LIST-OF-OPR indicates that S1 and S2 must be processed by the conversion routine after they have been translated.

Due to the lack of selectivity provided by the TABLE-OF-CONNECTIONS, a desirability filter for FORM-OPERATORs was added to the REDUCE-METHOD before giving this task to GPS. To test the desirability of a FORM-OPERATOR, the FEATURE, which is the type of difference, is evaluated in the output form of the FORM-OPERATOR. If this value is the same as the value of the difference, the operator is considered desirable. The significance of this filter is discussed later.

Behavior of GPS

Fig. 76 illustrates how GPS solved the task in Fig. 73. In order to achieve TOP-GOAL, GPS attempts to reduce D1 on the TOP-NODE of INITIAL-OBJ (GOAL 2). GOAL 3 is created because S1 is considered desirable, i.e., the OBJECT-SCHEMA produced by an application of S1 has SENTENCE as the value of

```
1 TOP-GOAL TRANSFORM INITIAL-OBJ INTO DESIRED-OBJ      (SUBGOAL OF NONE)

  2 GOAL 2 REDUCE D1 ON INITIAL-OBJ      (SUBGOAL OF TOP-GOAL)

    3 GOAL 3 APPLY S1 TO INITIAL-OBJ      (SUBGOAL OF 2)

      4 GOAL 4 REDUCE D1 ON INITIAL-OBJ      (SUBGOAL OF 3)

        5 GOAL 5 APPLY N TO INITIAL-OBJ      (SUBGOAL OF 4)

          6 GOAL 6 REDUCE D1 ON INITIAL-OBJ      (SUBGOAL OF 5)

            7 GOAL 7 APPLY A1 TO INITIAL-OBJ      (SUBGOAL OF 6)

            7 GOAL 8 APPLY A2 TO INITIAL-OBJ      (SUBGOAL OF 6)
                    OBJECT 7: (ADJECTIVE-PHRASE VARIABLES CAUSE CONFUSION PERIOD)

          6 GOAL 9 APPLY N TO 7      (SUBGOAL OF 5)
                    OBJECT 8: (NOUN-PHRASE CAUSE CONFUSION PERIOD)

      4 GOAL 10 APPLY S1 TO 8      (SUBGOAL OF 3)

        5 GOAL 11 REDUCE D1 ON  NEXT 8      (SUBGOAL OF 10)

          6 GOAL 12 APPLY V1 TO  NEXT 8      (SUBGOAL OF 11)

          6 GOAL 13 APPLY V2 TO  NEXT 8      (SUBGOAL OF 11)
                    OBJECT 9: (NOUN-PHRASE VERB-PHRASE CONFUSION PERIOD)

        5 GOAL 14 APPLY S1 TO 9      (SUBGOAL OF 10)

          6 GOAL 15 REDUCE D1 ON  NEXT  NEXT 9      (SUBGOAL OF 14)

            7 GOAL 16 APPLY N TO  NEXT  NEXT 9      (SUBGOAL OF 15)

            7 GOAL 17 APPLY N1 TO  NEXT  NEXT 9      (SUBGOAL OF 15)
                    OBJECT 10: (NOUN-PHRASE VERB-PHRASE NOUN-PHRASE PERIOD)
```

Figure 76: The performance of GPS on the task in Fig. 73.

```
    6 GOAL 18 APPLY S1 TO 10        (SUBGOAL OF 14)
              OBJECT 11! (SENTENCE)

  2  GOAL 19 TRANSFORM 11 INTO DESIRED-OBJ        (SUBGOAL OF TOP-GOAL)

SUCCESS
```

Figure 76: (continued)

the FEATURE, D1. Since the value of PS at the TOP-NODE of INITIAL-OBJ is not NOUN-PHRASE (GOAL 4), GOAL 5 is created because N has the capability of alleviating this difference.

GPS attempts to apply A1 to INITIAL-OBJ (GOAL 6, GOAL 7), because the value of PS must be ADJECTIVE-PHRASE in order for N to be applicable. GPS notices that GOAL 7 is impossible because VARIABLES cannot be used as an ADJECTIVE. GOAL 8 is created in reattempting GOAL 6, and OBJECT 7 to which N can be applied is produced (GOAL 9, OBJECT 8). (This task uses a special routine for printing OBJECT-SCHEMAs in order to make them more legible. Only the values of ATTRIBUTEs are printed; LOC-PROGs and ATTRIBUTEs are not printed.)

S1 cannot be applied to OBJECT 8 (GOAL 10), because the value of PS of the NEXT is not VERB-PHRASE. An attempt to apply V1 to the NEXT of OBJECT 8 (GOAL 11, GOAL 12) fails because CAUSE cannot be used as an ADVERB. But the application of V2 to the NEXT of OBJECT 8 produces OBJECT 9 (GOAL 13). Finally, noticing that CONFUSION can be used as a NOUN-PHRASE (OBJECT 10), S1 is applied and, since its result (OBJECT 11) is identical to the DESIRED-OBJ, the task is solved.

Discussion

All of the operators of this task modify the value of PS at some node and thus are desirable to reducing the type of difference, D1. However, for a particular difference, at most two operators are desirable. For example, only S1 and S2 can alleviate the difference,

PS should be SENTENCE.

Only information about the types of differences and not about the values of differences can be put into the TABLE-OF-CONNECTIONS. Consequently, the

TABLE-OF-CONNECTIONS in Fig. 73 does not contain sufficient selectivity and an additional desirability filter for FORM-OPERATORs had to be given to GPS. On the other hand, the desirability-selection process for MOVE-OPERATORs gives GPS sufficient selectivity. If S1 and S2 (the only FORM-OPERATORs in Fig. 73) were expressed as MOVE-OPERATORs, the desirability filter would not need to be added. However, S1 and S2 were expressed as FORM-OPERATORs to demonstrate that the two different TYPEs of operators can be used in the specification of a single task.

A great deal of effort has been devoted to the construction of efficient parsing algorithms for simple phase structure grammars, Oettinger [47]. The point of this example is not GPS's proficiency as a parser, but to illustrate the kinship between heuristic search and syntactic analysis.

I. BRIDGES OF KONIGSBERG

In the German town of Königsberg ran the river Pregel. In the river were two islands connected with the mainland and with each other by seven bridges as shown in Fig. 77. Is it possible for a person to walk from some point in the town and return to the same point after crossing each of the seven bridges once and only once?

In 1736 Euler proved that this task is impossible, and his proof stands as one of the early efforts in topology, Northrop [46]. However, we can give GPS the task of finding a path which starts at point E in Fig. 77, crosses all of the bridges precisely once and ends at point E, even though we know a priori that a solution does not exist.

GPS Formulation

An OBJECT-SCHEMA in the formulation in Fig. 78 is a single node with

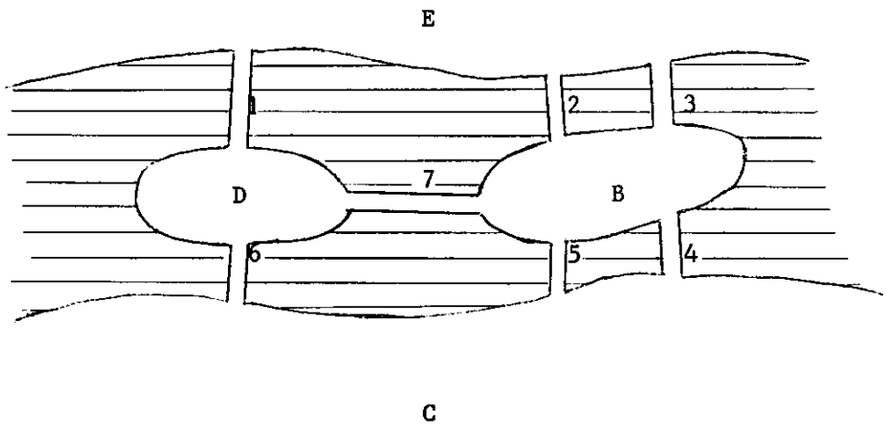


FIGURE 77. A schematic of the seven bridges of Königsberg. The numbers, 1, 2, ..., 7, are labels for the bridges, and the letters, B, C, D, E, are labels for the different sectors of town.

```
DECLARE
   BRIDGES = SET
   BRIDGE-1 = ATTRIBUTE
   BRIDGE-2 = ATTRIBUTE
   BRIDGE-3 = ATTRIBUTE
   BRIDGE-4 = ATTRIBUTE
   BRIDGE-5 = ATTRIBUTE
   BRIDGE-6 = ATTRIBUTE
   BRIDGE-7 = ATTRIBUTE
   CROSS = MOVE-OPERATOR
   CURRENT-POINT = ATTRIBUTE
   DESIRED-OBJ = DESCRIBED-OBJ
   D1 = FEATURE
   D2 = FEATURE
   D3 = FEATURE
   D4 = FEATURE
   D5 = FEATURE
   D6 = FEATURE
   D7 = FEATURE
   D8 = FEATURE
   ED = SET
   EB = SET
   CB = SET
   CD = SET
   DB = SET
   ENDS = ATTRIBUTE
   INITIAL-OBJ = OBJECT-SCHEMA
```

Figure 78: The specification for GPS of the bridges of Königsberg.

```
   X = ATTRIBUTE
   )
TASK-STRUCTURES   (
   OBJ-ATTRIB = ( BRIDGE-1 BRIDGE-2 BRIDGE-3 BRIDGE-4 BRIDGE-5
                  BRIDGE-6 BRIDGE-7 CURRENT-POINT )
   TOP-GOAL = ( TRANSFORM THE INITIAL-OBJ INTO THE DESIRED-OBJ . )
   INITIAL-OBJ = ( CURRENT-POINT E )
   DESIRED-OBJ = ( TEX-DESCRIPTION
                   1. X EQUALS CROSSED , FOR-ALL X IN THE BRIDGES .
                   2. THE CURRENT-POINT EQUALS E .    )
   BRIDGES = ( BRIDGE-1 BRIDGE-2 BRIDGE-3 BRIDGE-4 BRIDGE-5 BRIDGE-6 BRIDGE-7 )
   CROSS = ( CREATION-OPERATOR
             VAR-DOMAIN
             1. OTHER-END IS AN EXCLUSIVE-MEMBER OF THE ENDS OF THE
                PARTICULAR X .
             2. NEXT-POINT IS AN EXCLUSIVE-MEMBER OF THE ENDS OF THE
                PARTICULAR X .
             3. X IS IN-THE-SET OF BRIDGES .
             PRETESTS
             1. CURRENT-POINT EQUALS THE OTHER-END .
             2. X IS UNDEFINED .
             MOVES
             1. COPY CROSSED AT X .
             2. COPY NEXT-POINT AT THE CURRENT-POINT . )
   BRIDGE-1 = ( ENDS ED )
   BRIDGE-2 = ( ENDS EB )
   BRIDGE-3 = ( ENDS EB )
```

Figure 78: (continued)

```
    BRIDGE-4 = ( ENDS CB )
    BRIDGE-5 = ( ENDS CB )
    BRIDGE-6 = ( ENDS CD )
    BRIDGE-7 = ( ENDS DB )
    ED = ( E D )
    EB = ( E B )
    CB = ( C B )
    CD = ( C D )
    DB = ( D B )
    D1 = ( BRIDGE-1 )
    D2 = ( BRIDGE-2 )
    D3 = ( BRIDGE-3 )
    D4 = ( BRIDGE-4 )
    D5 = ( BRIDGE-5 )
    D6 = ( BRIDGE-6 )
    D7 = ( BRIDGE-7 )
    D8 = ( CURRENT-POINT )
    DIFF-ORDERING = ( 1. ( D1 D2 D3 D4 D5 D6 D7 )
                     2. D8  )
    TABLE-OF-CONNECTIONS = ( ( COMMON-DIFFERENCE CROSS ) )
    LIST-OF-VAR = ( X NEXT-POINT OTHER-END )
    )
END
```

Figure 78: (continued)

up to eight ATTRIBUTEs (OBJ-ATTRIB). The value of CURRENT-POINT is either B, C, D, or E corresponding to where GPS would be standing in Fig. 77. The values of the other ATTRIBUTEs indicate which bridges have been crossed. (BRIDGE-1 corresponds to the bridge labeled 1 in Fig. 77; BRIDGE-2 corresponds to the bridge labeled 2; etc.) For example, BRIDGE-1 has been crossed if the value of the ATTRIBUTE, BRIDGE-1, is CROSSED; otherwise, BRIDGE-1 has not been crossed.

TOP-GOAL is the statement of the task. INITIAL-OBJ represents the situation when GPS is standing at point E and has not crossed any bridges.

The DESIRED-OBJ is a DESCRIBED-OBJ which represents the situation when all of the bridges have been CROSSED and GPS is standing at point E. The first TEST in the DESIRED-OBJ requires that all of the ATTRIBUTEs in the SET, BRIDGES, have the value CROSSED. And the second TEST is satisfied if CURRENT-POINT has the value E.

The only operator in this task is CROSS whose application corresponds to walking from the point, OTHER-END, across the bridge, X, to the point, NEXT-POINT. The third TEST in VAR-DOMAIN requires that X is in the SET, BRIDGES. Each member of BRIDGES is not an atomic symbol, but a data structure that is an encoding of the two points connected by the bridge, e.g., BRIDGE-1 connects E and D. The first two TESTs in VAR-DOMAIN signify that OTHER-END and NEXT-POINT stand for the two points connected by the bridge, X.

The PRETESTS indicate that in order to cross the bridge, X, GPS must be standing at one end of X and that GPS has not previously crossed X. The MOVES designate that in the resultant object X is CROSSED and the value of CURRENT-POINT is NEXT-POINT. (According to the PRETESTS, the CURRENT-POINT must be OTHER-END in the object to which CROSS is applied.)

In the formulation in Fig. 78 of this task, there are eight differences, D1...D8, each referring to the value of one of the eight ATTRIBUTEs. DIFF-ORDERING indicates that these differences that refer to the status of a bridge, D1...D7, are more difficult than D8 which refers to the point where GPS would be standing in Fig. 77. TABLE-OF-CONNECTIONS designates that CROSS is relevant to reducing any type of difference and LIST-OF-VAR lists the variables which appear in Fig. 78.

GPS often detects several of the types of differences, D1, D2,...D7, between two objects. Since they are all equally difficult, GPS must arbitrarily select one and attempt to reduce it. For this reason, the TRANSFORM-METHOD was slightly generalized for this task. If an attempt to reduce a difference, which is detected by the TRANSFORM-METHOD, fails, the method does not necessarily fail. Instead, another equally difficult difference will be selected and the GOAL of reducing this difference will be generated. This modified TRANSFORM-METHOD was also used in the missionaries and cannibals task and is discussed in more detail on pages 171-2.

Behavior of GPS

Fig. 79 illustrates how GPS attempted to solve the task in Fig. 78. In attempting TOP-GOAL, GPS detects the differences--D1...D7--and since they are all equally difficult, GPS selects one, D7, to REDUCE (GOAL 2). GPS attempts to apply CROSS with X equal to BRIDGE-7 because it is relevant to reducing D7 on INITIAL-OBJ. GPS could CROSS BRIDGE-7 by walking from B to D, if CURRENT-POINT had the value B instead of E (GOAL 4). After crossing BRIDGE-2 (GOAL 5, OBJECT 4), BRIDGE-7 is CROSSED (GOAL 6, OBJECT 5).

Since BRIDGE-6 is not CROSSED in OBJECT 5 (GOAL 7, GOAL 8), GPS crosses it (GOAL 9, OBJECT 6). GPS attempts to CROSS BRIDGE-5 from B to C

```
1 TOP-GOAL: TRANSFORM INITIAL-OBJ INTO DESIRED-OBJ      (SUBGOAL OF NONE)

  2 GOAL: 2 REDUCE D7 ON INITIAL-OBJ      (SUBGOAL OF TOP-GOAL)

    3 GOAL: 3 APPLY CROSS WITH X = BRIDGE-7, TO INITIAL-OBJ      (SUBGOAL OF 2)
                  SET: OTHER-END = B, NEXT-POINT = D

      4 GOAL: 4 REDUCE D8 ON INITIAL-OBJ      (SUBGOAL OF 3)

        5 GOAL: 5 APPLY CROSS WITH NEXT-POINT = B, TO INITIAL-OBJ      (SUBGOAL OF 4)
                  SET: OTHER-END = E, X = BRIDGE-2
                  OBJECT 4: (BRIDGE-2 CROSSED CURRENT-POINT B)

      4 GOAL: 6 APPLY CROSS WITH X = BRIDGE-7, TO 4      (SUBGOAL OF 3)
                  SET: OTHER-END = B, NEXT-POINT = D
                  OBJECT 5: (BRIDGE-2 CROSSED BRIDGE-7 CROSSED CURRENT-POINT D)

  2 GOAL: 7 TRANSFORM 5 INTO DESIRED-OBJ      (SUBGOAL OF TOP-GOAL)

    3 GOAL: 8 REDUCE D6 ON 5      (SUBGOAL OF 7)

      4 GOAL: 9 APPLY CROSS WITH X = BRIDGE-6, TO 5      (SUBGOAL OF 8)
                  SET: OTHER-END = D, NEXT-POINT = C
                  OBJECT 6: (BRIDGE-2 CROSSED BRIDGE-6 CROSSED BRIDGE-7 CROSSED CURRENT-POINT C)

    3 GOAL: 10 TRANSFORM 6 INTO DESIRED-OBJ      (SUBGOAL OF 7)

      4 GOAL: 11 REDUCE D5 ON 6      (SUBGOAL OF 10)

        5 GOAL: 12 APPLY CROSS WITH X = BRIDGE-5, TO 6      (SUBGOAL OF 11)
                  SET: OTHER-END = B, NEXT-POINT = C

          6 GOAL: 13 REDUCE D8 ON 6      (SUBGOAL OF 12)

            7 GOAL: 14 APPLY CROSS WITH NEXT-POINT = B, TO 6      (SUBGOAL OF 13)
                  SET: OTHER-END = C, X = BRIDGE-4
                  OBJECT 7: (BRIDGE-2 CROSSED BRIDGE-4 CROSSED BRIDGE-6 CROSSED BRIDGE-7 CROSSED
                        CURRENT-POINT B)

          6 GOAL: 15 APPLY CROSS WITH X = BRIDGE-5, TO 7      (SUBGOAL OF 12)
                  SET: OTHER-END = B, NEXT-POINT = C
```

Figure 79: The performance of GPS on the bridges of Königsberg.

```
                OBJECT 8: (BRIDGE-2 CROSSED BRIDGE-4 CROSSED BRIDGE-5 CROSSED BRIDGE-6 CROSSED
                          BRIDGE-7 CROSSED CURRENT-POINT C)

4  GOAL 16 TRANSFORM 8 INTO DESIRED-OBJ      (SUBGOAL OF 10)

  5 GOAL 17 REDUCE D3 ON 8      (SUBGOAL OF 16)

    6 GOAL 18 APPLY CROSS WITH X = BRIDGE-3, TO 8      (SUBGOAL OF 17)
              SET: OTHER-END = B, NEXT-POINT = E

      7 GOAL 19 REDUCE D8 ON 8      (SUBGOAL OF 18)

        8 GOAL 20 APPLY CROSS WITH NEXT-POINT = B, TO 8      (SUBGOAL OF 19)
              SET: OTHER-END = B, X = BRIDGE-7

  5 GOAL 21 REDUCE D1 ON 8      (SUBGOAL OF 16)

    6 GOAL 22 APPLY CROSS WITH X = BRIDGE-1, TO 8      (SUBGOAL OF 21)
              SET: OTHER-END = D, NEXT-POINT = E

      7 GOAL 23 REDUCE D8 ON 8      (SUBGOAL OF 22)

        8 GOAL 24 APPLY CROSS WITH NEXT-POINT = D, TO 8      (SUBGOAL OF 23)
              SET: OTHER-END = B, X = BRIDGE-7

4 GOAL 11 REDUCE D5 ON 6      (SUBGOAL OF 10)

4 GOAL 25 REDUCE D4 ON 6      (SUBGOAL OF 10)

  5 GOAL 26 APPLY CROSS WITH X = BRIDGE-4, TO 6      (SUBGOAL OF 25)
              SET: OTHER-END = B, NEXT-POINT = C

    6 GOAL 27 REDUCE D6 ON 6      (SUBGOAL OF 26)

4 GOAL 29 REDUCE D3 ON 6      (SUBGOAL OF 10)

  5 GOAL 30 APPLY CROSS WITH X = BRIDGE-3, TO 6      (SUBGOAL OF 29)
              SET: OTHER-END = B, NEXT-POINT = E

    6 GOAL 31 REDUCE D8 ON 6      (SUBGOAL OF 30)
```

Figure 79: (continued)

```
  4 GOAL 33 REDUCE D1 ON 6      (SUBGOAL OF 10)

    5 GOAL 34 APPLY CROSS WITH X = BRIDGE-1, TO 6      (SUBGOAL OF 33)
                SET: OTHER-END = D, NEXT-POINT = E

      6 GOAL 35 REDUCE D8 ON 6      (SUBGOAL OF 34)

        7 GOAL 36 APPLY CROSS WITH NEXT-POINT = D, TO 6      (SUBGOAL OF 35)
                SET: OTHER-END = B, X = BRIDGE-7

3 GOAL 8 REDUCE D6 ON 5      (SUBGOAL OF 7)

3 GOAL 37 REDUCE D5 ON 5      (SUBGOAL OF 7)

  4 GOAL 38 APPLY CROSS WITH X = BRIDGE-5, TO 5      (SUBGOAL OF 37)
                SET: OTHER-END = B, NEXT-POINT = C

    5 GOAL 39 REDUCE D8 ON 5      (SUBGOAL OF 38)

      6 GOAL 40 APPLY CROSS WITH NEXT-POINT = B, TO 5      (SUBGOAL OF 39)
                SET: OTHER-END = B, X = BRIDGE-7

3 GOAL 41 REDUCE D4 ON 5      (SUBGOAL OF 7)

  4 GOAL 42 APPLY CROSS WITH X = BRIDGE-4, TO 5      (SUBGOAL OF 41)
                SET: OTHER-END = B, NEXT-POINT = C

    5 GOAL 43 REDUCE D8 ON 5      (SUBGOAL OF 42)

3 GOAL 45 REDUCE D3 ON 5      (SUBGOAL OF 7)

  4 GOAL 46 APPLY CROSS WITH X = BRIDGE-3, TO 5      (SUBGOAL OF 45)
                SET: OTHER-END = B, NEXT-POINT = E

    5 GOAL 47 REDUCE D8 ON 5      (SUBGOAL OF 46)

3 GOAL 49 REDUCE C1 ON 5      (SUBGOAL OF 7)
```

Figure 79: (continued)

```
4 GOAL 50 APPLY CROSS WITH X = BRIDGE-1, TO 5      (SUBGOAL OF 49)
          SET: OTHER-END = D, NEXT-POINT = E
          OBJECT 9: (BRIDGE-1 CROSSED BRIDGE-2 CROSSED BRIDGE-7 CROSSED CURRENT-POINT E)

3 GOAL 51 TRANSFORM 9 INTO DESIRED-OBJ      (SUBGOAL OF 7)

  4 GOAL 52 REDUCE D6 ON 9      (SUBGOAL OF 51)

    5 GOAL 53 APPLY CROSS WITH X = BRIDGE-6, TO 9      (SUBGOAL OF 52)
          SET: OTHER-END = D, NEXT-POINT = C

      6 GOAL 54 REDUCE D8 ON 9      (SUBGOAL OF 53)

        7 GOAL 55 APPLY CROSS WITH NEXT-POINT = D, TO 9      (SUBGOAL OF 54)
          SET: OTHER-END = B, X = BRIDGE-7

  4 GOAL 56 REDUCE D5 ON 9      (SUBGOAL OF 51)

    5 GOAL 57 APPLY CROSS WITH X = BRIDGE-5, TO 9      (SUBGOAL OF 56)
          SET: OTHER-END = B, NEXT-POINT = C

      6 GOAL 58 REDUCE D8 ON 9      (SUBGOAL OF 57)

        7 GOAL 59 APPLY CROSS WITH NEXT-POINT = B, TO 9      (SUBGOAL OF 58)
          SET: OTHER-END = E, X = BRIDGE-3
          OBJECT 10: (BRIDGE-1 CROSSED BRIDGE-2 CROSSED BRIDGE-3 CROSSED BRIDGE-7 CROSSED
                      CURRENT-POINT B)

      6 GOAL 60 APPLY CROSS WITH X = BRIDGE-5, TO 10      (SUBGOAL OF 57)
          SET: OTHER-END = B, NEXT-POINT = C
          OBJECT 11: (BRIDGE-1 CROSSED BRIDGE-2 CROSSED BRIDGE-3 CROSSED BRIDGE-5 CROSSED
                      BRIDGE-7 CROSSED CURRENT-POINT C)

  4 GOAL 61 TRANSFORM 11 INTO DESIRED-OBJ      (SUBGOAL OF 51)

    5 GOAL 62 REDUCE D6 ON 11      (SUBGOAL OF 61)

      6 GOAL 63 APPLY CROSS WITH X = BRIDGE-6, TO 11      (SUBGOAL OF 62)
          SET: OTHER-END = D, NEXT-POINT = C

        7 GOAL 64 REDUCE D8 ON 11      (SUBGOAL OF 63)
```

Figure 79: (continued)

```
      8 GOAL 65 APPLY CROSS WITH NEXT-POINT = D, TO 11      (SUBGOAL OF 64)
            SET:  OTHER-END = C, X = BRIDGE-6
            OBJECT 12: (BRIDGE-1 CROSSED BRIDGE-2 CROSSED BRIDGE-3 CROSSED BRIDGE-5 CROSSED
                        BRIDGE-6 CROSSED BRIDGE-7 CROSSED CURRENT-POINT D)

    7 GOAL 66 APPLY CROSS WITH X = BRIDGE-6, TO 12      (SUBGOAL OF 63)
            SET:  OTHER-END = D, NEXT-POINT = D


    7 GOAL 64 REDUCE D8 ON 11      (SUBGOAL OF 63)


5 GOAL 67 REDUCE D4 ON 11      (SUBGOAL OF 61)


  6 GOAL 68 APPLY CROSS WITH X = BRIDGE-4, TO 11      (SUBGOAL OF 67)
            SET:  OTHER-END = B, NEXT-POINT = C


    7 GOAL 69 REDUCE D8 ON 11      (SUBGOAL OF 68)


      8 GOAL 70 APPLY CROSS WITH NEXT-POINT = B, TO 11      (SUBGOAL OF 69)
            SET:  OTHER-END = C, X = BRIDGE-4
            OBJECT 13: (BRIDGE-1 CROSSED BRIDGE-2 CROSSED BRIDGE-3 CROSSED BRIDGE-4 CROSSED
                        BRIDGE-5 CROSSED BRIDGE-7 CROSSED CURRENT-POINT B)

    7 GOAL 71 APPLY CROSS WITH X = BRIDGE-4, TO 13      (SUBGOAL OF 68)
```

Figure 79: (continued)

(GOAL 10, GOAL 11, GOAL 12) but cannot because the CURRENT-POINT in OBJECT 6 is C instead of B. Each bridge can be CROSSED in two directions but GPS does not realize that BRIDGE-5 can be CROSSED in the other direction. To make the CURRENT-POINT B (GOAL 13) BRIDGE-4 is CROSSED (GOAL 14, OBJECT-7). BRIDGE-5 can be CROSSED in OBJECT-7 (GOAL 15) which produces OBJECT-8.

Since none of the bridges can be CROSSED in OBJECT-8, the attempts to CROSS BRIDGE-3 (GOAL 18) and BRIDGE-1 (GOAL 22) both fail. GPS reattempts GOAL 11 to no avail because all of the desirable operators have been tried. Another attempt to achieve GOAL 10 generates GOAL 25 because D4 is as difficult as D5. But attempting GOAL 25 eventually leads to the generation of a GOAL, identical to GOAL 14, which does not get retried. (This is the reason GOAL 27 is abandoned.) All other attempts to achieve GOAL 10 fail, because GPS fails to reduce D3 on OBJECT 6 (GOAL 29) and fails to reduce D1 on OBJECT 6 (GOAL 33).

In reattempting GOAL 7, GPS eventually produces OBJECT 9 (GOAL 50). GPS manages to cross six of the seven bridges twice (OBJECT 12, OBJECT 13) but shortly thereafter exhausts its memory.

Discussion

Although GPS was given the problem of starting from and returning to point E, by slightly reformulating INITIAL-OBJ and DESIRED-OBJ, Fig. 78 would specify the problem of starting from and returning to an arbitrary point. The revised INITIAL-OBJ would be

(CURRENT-POINT X STARTING-POINT X)

and the DESIRED-OBJ would check if the CURRENT-POINT EQUALS the STARTING-POINT instead of point E. X would be bound by applying an operator to the INITIAL-OBJ.

Eventually, GPS would give up on this task, because it would run out of things to do. But GPS would have tried all possibilities before it ran out of things to do, even though GPS would not realize it had disproved the problem by exhaustion. In attempting a TRANSFORM-GOAL, a difference is generated for each bridge which is not CROSSED. The modified TRANSFORM-METHOD will generate a REDUCE-GOAL for each such difference and GPS will attempt to apply an operator that crosses the bridge to which the difference pertains.

In attempting most impossible tasks, GPS would not attempt to search the entire problem space because some parts of the space would appear undesirable. However, in this task, most GOALs appear equally desirable because the difference types, D1, ...D7 are all equally difficult according to DIFF-ORDERING. A better DIFF-ORDERING and a better set of difference types would increase GPS's selectivity. But even if GPS could use more sophisticated types of differences (than the FEATUREs used currently), it is not clear what types of differences should be used to increase GPS's selectivity.

The impossibility of this problem lies in the topological properties of the city and by studying these properties, Euler discovered the problem to be impossible. GPS cannot see the impossibility because it sets out to CROSS bridges instead of discovering topological properties of the city. Such limitations in problem solving programs are discussed in detail in Newell [37].

J. WATER JUG TASK

Given a five gallon jug and an eight gallon jug, how can precisely two gallons be put into the five gallon jug? Since there is a sink nearby, a jug can be filled from the tap and can be emptied by pouring its contents

down the drain. Water can be poured from one jug into another, but no measuring devices are available other than the jugs themselves[11].

This is only a particular water jug task. In others, the number of jugs, the size of the jugs, and the amount of water desired may be different, but the general problem is the same. Sometimes, as an additional constraint, none of the water can be poured down the drain[12]. This task has been used extensively in psychological experiments designed to investigate certain aspects of human behavior, Luchins [25].

GPS Formulation

Fig. 80 is the formulation of the task for GPS. INITIAL-OBJ, an OBJECT-SCHEMA whose tree structure representation is shown in Fig. 81, is the situation when both the five gallon jug and the eight gallon jug are empty. A jug is represented by a node which has three ATTRIBUTEs, listed in OBJ-ATTRIB:

> MAXIMUM whose value is the size of the jug;
> CONTENTS whose value is the amount of water in the jug;
> SPACE whose value is the size of the jug minus the amount of water in the jug.

SPACE is clearly superfluous, but it makes the statement of the operators simpler.

FINAL-OBJ consists of a single TEST to be applied at the TOP-NODE of an OBJECT-SCHEMA. FINAL-OBJ represents the situation when there are two gallons in the five gallon jug and TOP-GOAL is the statement of the problem.

---

[11]The source of this problem is #19 in Mott-Smith [32].

[12]Given a three liter jug and a five liter jug, both of which are empty, and given an eight liter jug full of ale, how can two men split the ale? In this task, a jug cannot be filled from the tap and, of course, none of the ale can be poured down the drain.

```
SKIP-WORDS    (
   GALLONS
     )
RENAME    (
   JUG-1 = FIRST
   JUG-2 = SECOND
     )
DECLARE    (
   FINAL-OBJ = DESCRIBED-OBJ
   INITIAL-OBJ = OBJECT-SCHEMA
   EMPTY-JUG = MOVE-OPERATOR
   FILL-JUG = MOVE-OPERATOR
   CREATE-WATER = MOVE-OPERATOR
   DESTROY-WATER = MOVE-OPERATOR
   D1-1 = FEATURE
   D1-2 = FEATURE
   D2-1 = FEATURE
   D2-2 = FEATURE
   CONTENTS = ATTRIBUTE
   MAXIMUM = ATTRIBUTE
   SPACE = ATTRIBUTE
   JUGS = SET
   NUMBERS = SET
   X = LOC-PROG
   Y = LOC-PROG
     )
TASK-STRUCTURES    (
```

Figure 80: The specification for GPS of a water jug task.

```
INITIAL-OBJ  = ( JUG-1 ( CONTENTS  IS 0   SPACE IS 5 GALLONS
                         MAXIMUM IS 5 GALLONS )
                JUG-2 ( CONTENTS IS 0  SPACE IS 8 GALLONS
                         MAXIMUM IS 8 GALLONS ) )
OBJ-ATTRIB = ( CONTENTS SPACE MAXIMUM )
FINAL-OBJ = ( THE TEX-DESCRIPTION IS THE CONTENTS OF JUG-1 EQUALS 2
              GALLONS . )
TOP-GOAL = ( TRANSFORM THE INITIAL-OBJ INTO THE FINAL-OBJ . )
CREATE-WATER = (  $ FILL JUG X WITH WATER FROM THE TAP $
                  CREATION-OPERATOR
                  VAR-DOMAIN
                  X IS A CONSTRAINED-MEMBER OF THE JUGS , THE CONSTRAINT
                  IS THAT THE CONTENTS OF X DOES NOT-EQUAL THE MAXIMUM OF
                  X .
                  MOVES
                  1. COPY THE MAXIMUM OF X AT THE CONTENTS OF X .
                  2. COPY 0 AT THE SPACE IN X .   )
DESTROY-WATER = ( $ POUR AWAY ALL OF THE WATER IN JUG X $
                  CREATION-OPERATOR
                  VAR-DOMAIN
                  X IS A CONSTRAINED-MEMBER OF THE JUGS , THE CONSTRAINT
                  IS THAT THE CONTENTS OF X DOES NOT-EQUAL 0
                  MOVES
                  1. COPY THE MAXIMUM OF X AT THE SPACE IN X .
                  2. COPY 0 AT THE CONTENTS OF X .   )
EMPTY-JUG = $ EMPTY ALL OF JUG X INTO JUG Y $
            ( CREATION-OPERATOR
```

Figure 80: (continued)

```
                VAR-DOMAIN
                1. X IS AN EXCLUSIVE-MEMBER OF THE JUGS .
                2. Y IS AN EXCLUSIVE-MEMBER OF THE JUGS .
                3. Z IS IN-THE-SET OF NUMBERS .
                PRETESTS
                1. THE CONTENTS OF X EQUALS Z .
                MOVES
                1. DELETE THE AMOUNT Z FROM THE SPACE IN Y AND
                   ADD IT TO THE CONTENTS OF Y .
                2. COPY 0 AT THE CONTENTS OF X .
                3. COPY THE MAXIMUM OF X AT THE SPACE IN X .    )
FILL-JUG = $ EMPTY PART OF JUG X INTO JUG Y $
           ( CREATION-OPERATOR
             VAR-DOMAIN
             1. X IS AN EXCLUSIVE-MEMBER OF THE JUGS .
             2. Y IS AN EXCLUSIVE-MEMBER OF THE JUGS .
             3. Z IS IN-THE-SET OF NUMBERS .
             PRETESTS
             1. THE SPACE IN Y EQUALS Z .
             MOVES
             1. DELETE THE AMOUNT Z FROM THE CONTENTS OF X
                AND ADD IT TO THE SPACE IN X .
             2. COPY THE MAXIMUM OF Y AT THE CONTENTS OF Y .
             3. COPY 0 AT THE SPACE IN Y .      )
NUMBERS = ( 1 2 3 4 5 )
JUGS = ( JUG-1 JUG-2 )
D1-1 = ( THE CONTENTS OF JUG-1 . )
D1-2 = ( THE SPACE IN JUG-1 . )
```

Figure 80: (continued)

```
D2-1 = ( THE CONTENTS OF JUG-2 . )
D2-2 = ( THE SPACE IN JUG-2 . )
DIFF-ORDERING = ( ( D1-1 D1-2 D2-1 D2-2 ) )
TABLE-OF-CONNECTIONS = ( ( COMMON-DIFFERENCE FILL-JUG EMPTY-JUG
                                              DESTROY-WATER CREATE-WATER ) )
LIST-OF-VAR = ( X Y Z )
  )
  END .
```

Figure 80: (continued)

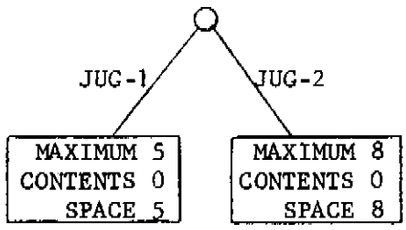


FIGURE 81. The tree structure representation of INITIAL-OBJ in Fig. 80.

There are four operators in this task. CREATE-WATER corresponds to filling a jug, X, from the tap. VAR-DOMAIN of CREATE-WATER requires that X is a jug which is not full. DESTROY-WATER corresponds to pouring the contents of jug, X, down the drain. According to its VAR-DOMAIN, X must be a jug which is not empty.

EMPTY-JUG and FILL-JUG are the operators for pouring Z gallons[13] of water from jug X into jug Y. Z must be in the SET, NUMBERS, i.e., an integer between 1 and 5 inclusively. Since the five gallon jug is always one of the jugs, Z cannot be more than 5 and it is never a fractional part of a gallon. An application of EMPTY-JUG corresponds to pouring all of the contents of jug X into jug Y. In order to do this the contents of jug X (Z) must fit into jug Y. If this is not the case, the first TRANSFORMATION, which adds the water to jug Y, will fail. The other two TRANSFORMATIONs have the effect of removing the water from jug X.

FILL-JUG is used to fill jug Y with water from the contents of jug X. The first TRANSFORMATION removes the water from jug X and will not be applicable unless there is more water in X than there is SPACE in Y. The other two TRANSFORMATIONs add the water to jug Y.

The types of differences for this task, D1-1, D1-2, D1-3, and D1-4, refer to the CONTENTS and SPACE of the two jugs. DIFF-ORDERING and TABLE-OF-CONNECTIONS are a mere formality for this task because, for lack of something better, all differences are considered equally difficult and all of the operators are considered desirable to reducing all types of differences. Thus, these data structures contain no information about the nature of the problem.

---

[13] Z, which stands for an amount of water, should not be confused with X and Y which stand for jugs instead of an amount of water.

LIST-OF-VAR indicates that X, Y, and Z are variables.

NEW-OBJ is a selection criterion given to GPS as an IPL-V structure and hence does not appear in Fig. 80. It is used to select from a SET of OBJECT-SCHEMAs those members which do not appear in the statement of any of the TRANSFORM GOALs generated thus far. This task and several others use this criterion in the TRANSFORM-SET-METHOD (see page 171 for a detailed discussion). The rationale is that a new GOAL can be created by transforming an object that fulfills this criterion into the desired situation.

Behavior of GPS

Fig. 82 shows how GPS solved the task in Fig. 80. The only difference between INITIAL-OBJ and FINAL-OBJ (TOP-GOAL) which GPS found is that the amount of water in the five gallon jug should be increased by two gallons. To reduce this difference (GOAL 2), GPS tries to apply FILL-JUG with Y being the five gallon jug (GOAL 3). This operator is considered desirable because it has the effect of increasing the amount of water in the five gallon jug even though, if successful, it would increase the amount of water by five gallons instead of two gallons. GOAL 3 is abandoned by GPS because, before the operator could be applied, the amount of water in the eight gallon jug must be increased by at least five gallons, which is as difficult as increasing the amount of water in the five gallon jug.

In reattempting GOAL 2, GPS tries to pour precisely two gallons of water into the five gallon jug from the eight gallon jug (GOAL 4). In order to do this, the eight gallon jug must contain precisely two gallons, and GOAL 4 is abandoned. GOAL 5 is created in another attempt to achieve GOAL 2 because CREATE-WATER with X equal to the five gallon jug has the desirable effect of increasing the amount of water in the five gallon jug.

```
1 TOP-GOAL TRANSFORM INITIAL-OBJ INTO FINAL-OBJ      (SUBGOAL OF NONE)

  2 GOAL 2 REDUCE D1-1 ON INITIAL-OBJ      (SUBGOAL OF TOP-GOAL)

    3 GOAL 3 APPLY FILL-JUG WITH Y = JUG-1, TO INITIAL-OBJ      (SUBGOAL OF 2)
                  SET: X = JUG-2, Z = 5

    3 GOAL 4 APPLY EMPTY-JUG WITH Z = 2, Y = JUG-1, TO INITIAL-OBJ      (SUBGOAL OF 2)
                  SET: X = JUG-2

    3 GOAL 5 APPLY CREATE-WATER WITH X = JUG-1, TO INITIAL-OBJ      (SUBGOAL OF 2)
                  OBJECT 4: (JUG-1(SPACE 0 CONTENTS 5 MAXIMUM 5) JUG-2(SPACE 8 CONTENTS 0 MAXIMUM
                     8))

  2  GOAL 6 TRANSFORM 4 INTO FINAL-OBJ      (SUBGOAL OF TOP-GOAL)

    3 GOAL 7 REDUCE D1-1 ON 4      (SUBGOAL OF 6)

      4 GOAL 8 APPLY FILL-JUG WITH Z = 3, X = JUG-1, TO 4      (SUBGOAL OF 7)
                  SET: Y = JUG-2

      4 GOAL 9 APPLY EMPTY-JUG WITH X = JUG-1, TO 4      (SUBGOAL OF 7)
                  SET: Y = JUG-2, Z = 5
                  OBJECT 5: (JUG-1(SPACE 5 CONTENTS 0 MAXIMUM 5) JUG-2(SPACE 3 CONTENTS 5 MAXIMUM
                     8))

    3  GOAL 10 TRANSFORM 5 INTO FINAL-OBJ      (SUBGOAL OF 6)

      4 GOAL 11 REDUCE D1-1 ON 5      (SUBGOAL OF 10)

        5 GOAL 12 APPLY FILL-JUG WITH Y = JUG-1, TO 5      (SUBGOAL OF 11)
                  SET: X = JUG-2, Z = 5
                  OBJECT 4: (JUG-1(SPACE 0 CONTENTS 5 MAXIMUM 5) JUG-2(SPACE 8 CONTENTS 0 MAXIMUM
                     8))

  2  GOAL 6 TRANSFORM 4 INTO FINAL-OBJ      (SUBGOAL OF TOP-GOAL)

    3  GOAL 14 TRANSFORM 3 INTO FINAL-OBJ      (SUBGOAL OF TOP-GOAL)

      4 GOAL 15 SELECT FROM 3 A/C NEW-OBJ OF FINAL-OBJ      (SUBGOAL OF 14)
```

Figure 82: The performance of GPS on the task in Fig. 80.

```
                NONE SELECTED
  2 GOAL 2 REDUCE D1-1 ON INITIAL-OBJ      (SUBGOAL OF TOP-GOAL)

    3  GOAL 14 TRANSFORM 3 INTO FINAL-OBJ      (SUBGOAL OF TOP-GOAL)

  2 GOAL 2 REDUCE D1-1 ON INITIAL-OBJ      (SUBGOAL OF TOP-GOAL)

    3 GOAL 7 REDUCE D1-1 ON 4      (SUBGOAL OF 6)

      4 GOAL 17 APPLY DESTROY-WATER WITH X = JUG-1, TO 4      (SUBGOAL OF 7)
                OBJECT INITIAL-OBJ: (JUG-1(SPACE 5 CONTENTS 0 MAXIMUM 5) JUG-2(SPACE 8 CONTENTS
                                    0 MAXIMUM 8))
1 TOP-GOAL TRANSFORM INITIAL-OBJ INTO FINAL-OBJ      (SUBGOAL OF NONE)

    3 GOAL 7 REDUCE D1-1 ON 4      (SUBGOAL OF 6)

  2 GOAL 2 REDUCE D1-1 ON INITIAL-OBJ      (SUBGOAL OF TOP-GOAL)

    3 GOAL 7 REDUCE D1-1 ON 4      (SUBGOAL OF 6)

  2 GOAL 2 REDUCE D1-1 ON INITIAL-OBJ      (SUBGOAL OF TOP-GOAL)

      4 GOAL 11 REDUCE D1-1 ON 5      (SUBGOAL OF 10)

        5 GOAL 19 APPLY EMPTY-JUG WITH Z = 2, Y = JUG-1, TO 5      (SUBGOAL OF 11)
                  SET: X = JUG-2

        5 GOAL 20 APPLY CREATE-WATER WITH X = JUG-1, TO 5      (SUBGOAL OF 11)
                  OBJECT 6: (JUG-1(SPACE 0 CONTENTS 5 MAXIMUM 5) JUG-2(SPACE 3 CONTENTS 5 MAXIMUM
                            8))
      4  GOAL 21 TRANSFORM 6 INTO FINAL-OBJ      (SUBGOAL OF 10)

        5 GOAL 22 REDUCE D1-1 ON 6      (SUBGOAL OF 21)

          6 GOAL 23 APPLY FILL-JUG WITH Z = 3, X = JUG-1, TO 6      (SUBGOAL OF 22)
                    SET: Y = JUG-2
                    OBJECT 7: (JUG-1(SPACE 3 CONTENTS 2 MAXIMUM 5) JUG-2(SPACE 0 CONTENTS 8 MAXIMUM
                              8))
```

Figure 82: (continued)

```
5  GOAL 24 TRANSFORM 7 INTO FINAL-OBJ      (SUBGOAL OF 21)
SUCCESS
```

Figure 82: (continued)

In OBJECT-4, the five gallon jug contains too much water (GOAL 6, GOAL 7) and GOAL 8 is generated in an attempt to pour three gallons out of the five gallon jug. OBJECT 5 (the result of GOAL 9) does not contain enough water (GOAL 10, GOAL 11) and the water in the eight gallon jug is poured into the five gallon jug (GOAL 12) which results in the previously generated OBJECT 4. Since the GOAL of transforming OBJECT 4 into FINAL-OBJ is not a new GOAL, GPS does not try to attempt it, but looks for something else to do.

GOAL 14 is created in an attempt to generate a new GOAL. OBJECT 3, which is generated internally, is the SET of all OBJECT-SCHEMAs derived from the INITIAL-OBJ. GOAL 15 and thus GOAL 14 fail because all of the OBJECT-SCHEMA in OBJECT 3 have been used in a TRANSFORM GOAL.

As a last resort, GPS tries previously generated GOALs in an attempt to produce new results. In reattempting GOAL 2, GPS finds that it has already tried all of the desirable operators. GOAL 14 and GOAL 2 are abandoned because all of the methods for achieving these GOALs have been exhausted[14].

In reattempting GOAL 7, GPS creates GOAL 17 because DESTROY-WATER with X equal to the five gallon jug decreases the contents of the five gallon jug. Unfortunately, GOAL 17 leads to an old object, INITIAL-OBJ, and an old GOAL, TOP-GOAL, and again GPS retries unfinished goals.

After fumbling a bit, GPS decides to retry GOAL 11. GOAL 19 is considered infeasible and GOAL 20 results in OBJECT 6 which has five gallons

---

[14]All GOALs which were selected to be tried get printed. Some of these GOALs never really get attempted because all of the methods have been tried to exhaustion.

in both jugs. After filling the eight gallon jug with the five gallon jug, (GOAL 23) GPS notices that it has solved the problem (GOAL 24).

Discussion

Often the jug which is to contain the desired quantity of water is not given. The task of producing two gallons in either jug can be specified by reformulating the FINAL-OBJ in Fig. 81 as

ONE OF THE TWO-GALLON-TESTS is TRUE.

And the TWO-GALLON-TESTS is

1. THE CONTENTS OF JUG-1 EQUALS 2.
2. THE CONTENTS OF JUG-2 EQUALS 2.

However, the current object-difference process is not sophisticated enough to produce a difference when a disjunctive set of TESTs (indicated by TRUE) is not satisfied[15].

The use of differences in this task seems to be a rather ineffective means of guiding the problem solving. None of the APPLY GOALs generate subgoals. If a difference is detected in applying an operator, it is always as difficult as the difference which the application of the operator is supposed to reduce, because all of the types of differences are considered equally difficult (see DIFF-ORDERING). In such cases, GPS rejects the variable specification produced by the feasibility-selection process for MOVE-OPERATORs.

A better set of types of differences and a better ordering on them

15There is no conceptual difficulty in generalizing the object-difference process. But the difference produced should be the easiest difference, because satisfying any TEST will do.

might improve GPS's performance on this task. However, even if GPS could use more sophisticated types of differences (than FEATURE), it is unclear what types of differences would improve the problem solving. GPS might need some additional problem solving mechanisms, e.g., planning, in order to be more proficient at this task.

K. LETTER SERIES COMPLETION

This task, which is found in aptitude tests, is to add the next few letters to a series of letters. Several computer programs for solving this task (and similar tasks) have been constructed (Pivar and Finkelstein [48]; Simon and Kotovsky [55]; Williams [65]). A simple example of this task is given below:

B C B D B E _ _

The two blanks ('_') indicate that the next two letters of the series must be supplied. The answer to this task is that the next two letters are B and F respectively. This answer is based on the hypothesis that all odd letters of the series are B's and the even letters of the series are in the order of their occurrence in the English alphabet.

The next few letters of the series are derived from some general description of the series, as in the example above. In general, there are many different series whose first several letters are those given in the task. For example, another continuation of the above sequence is:

B C B D B E B C B D B E B C ...

But the correct answer is based on the "simplest" description of the series where simplest description, for lack of a better definition, is defined as the description that most people would use to describe the series.

Williams' Formulation of the Task

Donald Williams has constructed a program (Williams [65]) that does aptitude test problems among which is the letter series completion task. The formulation of this task for GPS was adapted from his formulation. Williams' program is to do over thirty different types of aptitude test problems; however, we will only describe how it handles the letter series completion task.

Williams' program knows certain relations among the letters of an alphabet. Listed in the order of their simplicity the relations used by the program are (the examples are taken from the English alphabet):

a. same -- B is the "same" as B
b. next -- C is the "next" letter after B
c. next after next -- D is the "next after next" letter after B
d. predecessor -- A is the "predecessor" of B.

A relationship is a relation with its pair of operands. In an attempt to find a description of a letter series, the program assigns relations between pairs of letters in the series.

The simplicity of a relationship is determined mainly by the simplicity of its relation. If two relationships have identical relations, the simpler one is the one whose operands are separated by fewer letters in the series. For example, in the series above, the relationship--the first letter is the same as the third letter--is simpler than the relationship--the first letter is the same as the fifth letter. But the latter is simpler than the relationship--the next letter of the alphabet follows the first letter.

The program assigns a relationship by scanning the series from left to right. It always attempts to find a relationship between the current letter,

and a letter to the right of the current letter. The current letter is initially the first letter in the series and upon assigning a relationship between the current letter and another letter, the other letter becomes the new current letter. Thus the scan does not necessarily move uniformly through the series, but jumps ahead as it successfully establishes relationships of letters which are not adjacent in the series.

The remainder of the techniques used by Williams' program on this task will be illustrated by an example rather than described generally. In attempting to solve our example sequence (using the English alphabet), the program begins by looking for the simplest relationship between the current letter (first letter) and some other letter. According to the program's definition of simplicity, it assigns the relationship--the first letter is the same as the third letter--and looks for the simplest relationship between the new current letter (third letter) and some letter to the right of it. It then assigns the relationship--the third letter is the same as the fifth letter.

At this point, it notices that both relationships assigned are equivalent; i.e., the relation is same in both and there is one letter between both first and third and third and fifth. Consequently, it checks to see if the fifth letter, which is the current letter, is the same as the seventh letter. Since the seventh letter is a blank, it assigns 'B' to be the seventh letter and assigns the relationship--the seventh is the same as the fifth.

In checking if the ninth letter is the same as the seventh, it notices that there is no ninth letter and assigns the second letter to be the new current letter because it is the first letter not previously used in a relationship. The simplest relationship--the fourth letter in the series is the next of the alphabet after the second letter in the series--is assigned

and the current letter becomes the fourth letter. Again the simplest relationship--the sixth letter is next after the fourth letter--is assigned. Since the last two relationships are the same, the program checks to see if the eighth letter is the next after the sixth. Since the eighth letter is a blank, 'F' is assigned to be the eighth letter of the series and the problem is solved.

The way Williams' program approaches this problem might seem strange because it assigns relationships until it stumbles across the answer[16]. Never does it obtain a description of the entire series, e.g., the odd letters are B's, etc., and check to see if the description correctly describes the letters given[17]. Nevertheless, Williams' program is quite proficient at the letter series completion task.

GPS Formulation

The transcription of the task,

B C B D B E _ _,

into the external representation of GPS given in Fig. 83, is considerably less direct than the other tasks we have discussed. The main reason is that the objects are not different letter series, but are partial descriptions of letter series. That is, an object is a series together with relationships on the various letters in the series.

Each node of an OBJECT-SCHEMA represents a position in a series. The TOP-NODE represents the first position of the series; the node, ONE (the new

16It does not always assign the correct relationships, and in such cases finds it necessary to go back and try different relationships.

17This technique is used in Simon and Kotovsky [55].

```
RENAME   (
   ONE = FIRST
   TWO = FIRST-FIRST
   THREE = FIRST-FIRST-FIRST
    )
DECLARE  (
   DESIRED-OBJ = DESCRIBED-OBJ
   APPLY-CURRENT-RELATION = MOVE-OPERATOR
   APPLY-LAST = MOVE-OPERATOR
   APPLY-RELATION = MOVE-OPERATOR
   CURRENT-RELATION = ATTRIBUTE
   INTERVAL = ATTRIBUTE
   LETTER = ATTRIBUTE
   RELATION = ATTRIBUTE
   D1 = FEATURE
   D2 = FEATURE
   QUOTE-Y = FEATURE
   INTERVALS = SET
   RELATIONS = SET
     )
LIST  (
  INITIAL-OBJ = ( B C B D B E - - )
     )
TASK-STRUCTURES   (
   TOP-GOAL = ( TRANSFORM THE INITIAL-OBJ INTO THE DESIRED-OBJ ; )
   DESIRED-OBJ = ( SUBEXPRESSION-TESTS
                   1. A RELATION IS DEFINED ;
```

Figure 83: The specification for GPS of the task of completing a letter series.

```
                2. THE CURRENT-RELATION IS UNDEFINED .   )
APPLY-RELATION = ( CREATION-OPERATOR
                   VAR-DOMAIN
                   1. Y IS IN-THE-SET OF INTERVALS .
                   2. X IS IN-THE-SET OF RELATIONS .
                   PRETESTS
                   1. THE LETTER OF Y EQUALS THE FUNCTION X OF THE LETTER .
                   MOVES
                   1. COPY X AT THE RELATION .
                   2. COPY X AT THE CURRENT-RELATION OF Y .
                   3.  COPY QUOTE-Y IN THE INTERVAL OF Y .
                   4.  COPY QUOTE-Y IN THE INTERVAL .
                      )
APPLY-CURRENT-RELATION = ( CREATION-OPERATOR
                           VAR-DOMAIN
                           1. X EQUALS THE CURRENT-RELATION  .
                           2. Y EQUALS THE INTERVAL .
                           PRETESTS
                           1. THE LETTER OF Y EQUALS THE FUNCTION X OF THE
                              LETTER .
                           MOVES
                           1. COPY THE CURRENT-RELATION AT THE RELATION .
                           2. MOVE THE CURRENT-RELATION TO THE
                              CURRENT-RELATION OF Y .
                           3. COPY QUOTE-Y IN THE INTERVAL OF Y .
                     )
APPLY-LAST = ( CREATION-OPERATOR
```

Figure 83: (continued)

```
                VAR-DOMAIN
                1. Y EQUALS THE INTERVAL .
                PRETESTS
                1. THE LETTER OF Y IS UNDEFINED .
                MOVES
                1. REMOVE THE CURRENT-RELATION .
                2. COPY LAST AT THE RELATION .      )
  INTERVALS = ( ONE TWO THREE )
  RELATIONS = ( SAME NEXT  )
  DIFF-ORDERING = ( D1 D2 )
  D1 = ( CURRENT-RELATION )
  D2 = ( RELATION )
  TABLE-OF-CONNECTIONS = ( 1. ( D1 APPLY-CURRENT-RELATION APPLY-LAST )
                           2. ( D2 APPLY-RELATION ) )
  NEXT = ( B C C D D E E F   )
  SAME = ( B B C C D D E E F F   )
  QUOTE-Y = ( QUOTE Y )
  OBJ-ATTRIB = ( LETTER CURRENT-RELATION RELATION INTERVAL )
  LIST-OF-VAR = ( X Y )
    )
END
```

Figure 83: (continued)

name of FIRST) from the TOP-NODE, represents the second position in the series; the node TWO from the TOP-NODE represents the third position in series, etc. Each node, except the last one, has precisely one branch leading from it to the node representing the following position in the series. The value of the ATTRIBUTE, LETTER, of a node is the letter in the position of the series represented by the node. Fig. 84 illustrates the tree structure representation of the INITIAL-OBJ--the series to be completed. In Fig. 84, α and β are variables, created by a special routine which converts INITIAL-OBJ in Fig. 83 into the tree structure in Fig. 84. Each blank must be replaced by a distinct variable, else GPS would consider that they all stand for the same letter.

A relationship on a pair of letters in a series is represented by the ATTRIBUTE, RELATION[18], at the node representing the left most letter of the pair. The relation (e.g., same) is the value of RELATION and the other letter of the relation is designated by the value of the ATTRIBUTE, INTERVAL. For example, the respective values, SAME and TWO, of the ATTRIBUTEs, RELATION and INTERVAL, at a node represent the relationship that the LETTER at the node is the SAME as the LETTER at the second position to the right.

TOP-GOAL is the statement of the problem. As in Williams' formulation, GPS does not set out to fill in the blanks. Rather, it attempts to assign relationships to the series filling in the blanks (substituting for the variables) whenever necessary. DESIRED-OBJ requires that all nodes have a value of the ATTRIBUTE, RELATION. Since noneof the relationships can involve

---

[18]RELATION is used as an ATTRIBUTE in this task and should not be confused with the RELATIONs of GPS, e.g., EQUALS and CONSTRAINED-MEMBER.

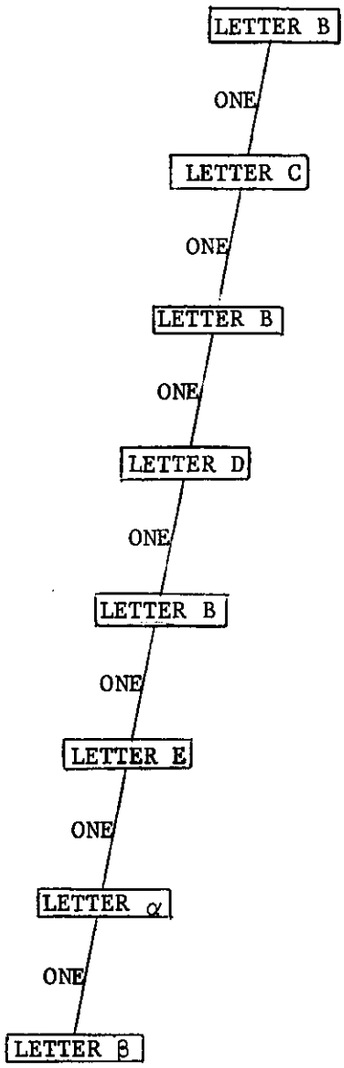


FIGURE 84. The tree structure representation of INITIAL-OBJ in Fig. 83 (α and β are variables).

a blank, this insures that all of the blanks have been filled.

The ATTRIBUTE, CURRENT-RELATION, of a node indicates that the node represents the current letter of the series. The second TEST in DESIRED-OBJ gives rise to a difference detected at a node that has a value of CURRENT-RELATION. In order to reduce this difference, a RELATION must be added to this node. The use of CURRENT-RELATION in this formulation corresponds to the fact that the current-letter is always a member of the pair of letters in a relationship assigned by Williams' program.

There are three operators, each of which assigns a relationship. APPLY-RELATION is used to assign the simplest relationship. It assumes that the current letter is the point of application. X is the relation of the relationship being assigned. Y is the distance between the letters of the relationship. For example, if Y is ONE, the two letters are adjacent; if Y is TWO, one letter separates them, etc.

PRETESTS of APPLY-RELATION requires that the pair of letters satisfy the relation X. For example, if X is SAME and Y is TWO, PRETESTS will be satisfied in applying APPLY-RELATION to the TOP-NODE of INITIAL-OBJ because the first and third letters of the series are both B's.

The first and fourth TRANSFORMATIONs of APPLY-RELATION assign a relationship to an OBJECT-SCHEMA. They add the attribute-value pairs--RELATION X, INTERVAL Y--to the node at which APPLY-RELATION is applied.

The other two TRANSFORMATIONs serve the purpose of marking the new current letter and noting the relationship assigned by the operator at the new current letter.

APPLY-CURRENT-RELATION applies the previously assigned relationship at the current letter marked by the ATTRIBUTE, CURRENT-RELATION. This operator is the same as APPLY-RELATION except that the legitimate values of

the variables are more restricted, and that the node to which it is applied must have a value of the ATTRIBUTE, CURRENT-RELATION. X and Y must have the values of the ATTRIBUTEs, CURRENT-RELATION and INTERVAL, respectively.

APPLY-LAST is used to handle the case when the letter, a distance, Y, from the current letter, does not exist, e.g., the ninth letter in INITIAL-OBJ. LAST is copied at the RELATION of the current letter to indicate that it has been considered. CURRENT-RELATION is removed so that the node at which the operator is being applied will no longer be considered the current letter.

DIFF-ORDERING ranks D1, which refers to the value of CURRENT-RELATION, as the more important type of difference, because it is detected only at a node that has a value of CURRENT-RELATION. Since such a node corresponds to Williams' current letter, the next operator must be applied at that node.

TABLE-OF-CONNECTIONS lists APPLY-CURRENT-RELATION and APPLY-LAST as desirable for reducing D1, because they affect CURRENT-RELATION. Even though they also affect RELATION, they are not considered desirable for reducing D2, which pertains to RELATION, because a GOAL of reducing D2 only is generated when the OBJECT-SCHEMA does not contain a CURRENT-RELATION.

NEXT is a list of pairs of letters. The first letter of each pair is a letter of the alphabet and the second letter of the pair is the following letter of the alphabet. It is used in the PRETESTS of APPLY-RELATION and APPLY-CURRENT-RELATION (a possible value of X) as a FUNCTION of one argument. For a particular value, α, of the argument, the value of the function is the second letter of the pair whose first letter is α. For example, NEXT of C is D.

SAME has the same role as NEXT. It is a list of pairs of letters and both letters in each pair are the same.

In some of the TRANSFORMATIONs of the operators, QUOTE-Y is an argument of an OPERATION. The value of the QUOTE-Y is always 'Y',e.g., ONE or TWO. Since Y is used as a LOC-PROG, normally the value of the argument, Y, is the node of an OBJECT-SCHEMA to which Y refers. Thus, QUOTE-Y had to be used instead of Y.

OBJ-ATTRIB is a list of all the ATTRIBUTEs for this task and LIST-OF-VAR is a list of the variables which appear in Fig. 83.

Behavior of GPS

Fig. 85 shows the behavior of GPS on the task in Fig. 83. In attempting TOP-GOAL, GPS notices that none of the nodes of INITIAL-OBJ has a RELATION and creates GOAL 2 to reduce this difference at the TOP-NODE (which corresponds to the left-most letter). To achieve GOAL 2, GPS attempts to apply APPLY-RELATION with none of its variables specified (GOAL 3) because, regardless of their values the operator is considered to be desirable.

GOAL 3 results in OBJECT 4 which has been assigned the relationship--the first letter is the same as the third letter. There is a special print routine for the OBJECT-SCHEMAs of this task. The positions in the series (nodes in the OBJECT-SCHEMAs) are separated by commas and no LOC-PROGs appear in the print-out of an OBJECT-SCHEMA. The TOP-NODE (left-most position of the series) of OBJECT 4 has three attribute-value pairs,

a. LETTER B
b. RELATION SAME
c. INTERVAL TWO.

The relationship that the second letter to the right is also a B corresponds to b and c. Henceforth, nodes of OBJECT-SCHEMAs will be referred to by their corresponding position in the series.

In OBJECT 4 the third position of the series, which is the only other position having an ATTRIBUTE other than LETTER, has the attribute-value pairs:

```
1 TOP-GOAL TRANSFORM INITIAL-OBJ INTO DESIRED-OBJ      (SUBGOAL OF NONE)

  2 GOAL 2 REDUCE D2 ON INITIAL-OBJ      (SUBGOAL OF TOP-GOAL)

    3 GOAL 3 APPLY APPLY-RELATION TO INITIAL-OBJ      (SUBGOAL OF 2)
                SET: Y = TWO, X = SAME
                OBJECT 4: (LETTER B RELATION SAME INTERVAL TWO, LETTER C, LETTER B CURRENT-RELATION
                          SAME INTERVAL TWO, LETTER D, LETTER B, LETTER E, LETTER -, LETTER -)

  2  GOAL 4 TRANSFORM 4 INTO DESIRED-OBJ      (SUBGOAL OF TOP-GOAL)

    3 GOAL 5 REDUCE D1 ON  ONE  ONE 4      (SUBGOAL OF 4)

      4 GOAL 6 APPLY APPLY-CURRENT-RELATION TO  ONE  ONE 4      (SUBGOAL OF 5)
                SET: X = SAME, Y = TWO
                OBJECT 5: (LETTER B RELATION SAME INTERVAL TWO, LETTER C, LETTER B RELATION SAME
                          INTERVAL TWO, LETTER D, LETTER B CURRENT-RELATION SAME INTERVAL TWO,
                          LETTER E, LETTER -, LETTER -)

    3  GOAL 7 TRANSFORM 5 INTO DESIRED-OBJ      (SUBGOAL OF 4)

      4 GOAL 8 REDUCE D1 ON  ONE  ONE  ONE  ONE 5      (SUBGOAL OF 7)

        5 GOAL 9 APPLY APPLY-CURRENT-RELATION TO  ONE  ONE  ONE  ONE 5      (SUBGOAL OF 8)
                SET: X = SAME, Y = TWO
                OBJECT 6: (LETTER B RELATION SAME INTERVAL TWO, LETTER C, LETTER B RELATION SAME
                          INTERVAL TWO, LETTER D, LETTER B RELATION SAME INTERVAL TWO, LETTER E
                          , LETTER B CURRENT-RELATION SAME INTERVAL TWO, LETTER -)

      4  GOAL 10 TRANSFORM 6 INTO DESIRED-OBJ      (SUBGOAL OF 7)

        5 GOAL 11 REDUCE D1 ON  ONE  ONE  ONE  ONE  ONE  ONE 6      (SUBGOAL OF 10)

          6 GOAL 12 APPLY APPLY-CURRENT-RELATION TO  ONE  ONE  ONE  ONE  ONE  ONE 6      (SUBGOAL OF 11)
                SET: X = SAME, Y = TWO

          6 GOAL 13 APPLY APPLY-LAST TO  ONE  ONE  ONE  ONE  ONE  ONE 6      (SUBGOAL OF 11)
                SET: Y = TWO
                OBJECT 7: (LETTER B RELATION SAME INTERVAL TWO, LETTER C, LETTER E RELATION SAME
                          INTERVAL TWO, LETTER D, LETTER B RELATION SAME INTERVAL TWO, LETTER E
                          , LETTER B RELATION LAST INTERVAL TWO, LETTER -)
```

Figure 85: The performance of GPS on the task specified in Fig. 83.

```
5  GOAL 14 TRANSFORM 7 INTO DESIRED-OBJ      (SUBGOAL OF 10)

  6 GOAL 15 REDUCE D2 ON  ONE 7      (SUBGOAL OF 14)

    7 GOAL 16 APPLY APPLY-RELATION TO  ONE 7      (SUBGOAL OF 15)
          SET: Y = TWO, X = NEXT
          OBJECT 8: (LETTER B RELATION SAME INTERVAL TWO, LETTER C RELATION NEXT INTERVAL
                     TWO, LETTER B RELATION SAME INTERVAL TWO, LETTER D CURRENT-RELATION NEXT
                     INTERVAL TWO, LETTER B RELATION SAME INTERVAL TWO, LETTER E, LETTER B
                     RELATION LAST INTERVAL TWO, LETTER -)

  6  GOAL 17 TRANSFORM 8 INTO DESIRED-OBJ      (SUBGOAL OF 14)

    7 GOAL 18 REDUCE D1 ON  ONE  ONE  ONE 8      (SUBGOAL OF 17)

      8 GOAL 19 APPLY APPLY-CURRENT-RELATION TO  ONE  ONE  ONE 8      (SUBGOAL OF 18)
          SET: X = NEXT, Y = TWO
          OBJECT 9: (LETTER B RELATION SAME INTERVAL TWO, LETTER C RELATION NEXT INTERVAL
                     TWO, LETTER B RELATION SAME INTERVAL TWO, LETTER D RELATION NEXT INTERVAL
                     TWO, LETTER B RELATION SAME INTERVAL TWO, LETTER E CURRENT-RELATION NEXT
                     INTERVAL TWO, LETTER B RELATION LAST INTERVAL TWO, LETTER -)

    7  GOAL 20 TRANSFORM 9 INTO DESIRED-OBJ      (SUBGOAL OF 17)

      9 GOAL 21 REDUCE D1 ON  ONE  ONE  ONE  ONE  ONE 9      (SUBGOAL OF 20)

        9 GOAL 22 APPLY APPLY-CURRENT-RELATION TO  ONE  ONE  ONE  ONE  ONE 9      (SUBGOAL OF 21)
            SET: X = NEXT, Y = TWO
            OBJECT 10: (LETTER B RELATION SAME INTERVAL TWO, LETTER C RELATION NEXT INTERVAL
                        TWO, LETTER B RELATION SAME INTERVAL TWO, LETTER D RELATION NEXT INTERVAL
                        TWO, LETTER B RELATION SAME INTERVAL TWO, LETTER E RELATION NEXT INTERVAL
                        TWO, LETTER B RELATION LAST INTERVAL TWO, LETTER F CURRENT-RELATION
                        NEXT INTERVAL TWO)

      8  GOAL 23 TRANSFORM 10 INTO DESIRED-OBJ      (SUBGOAL OF 20)

        9 GOAL 24 REDUCE D1 ON  ONE  ONE  ONE  ONE  ONE  ONE  ONE 10      (SUBGOAL OF 23)

          10 GOAL 25 APPLY APPLY-CURRENT-RELATION TO  ONE  ONE  ONE  ONE  ONE  ONE  ONE 10      (SUBGOAL OF 24
          )
          SET: X = NEXT, Y = TWO

          10 GOAL 26 APPLY APPLY-LAST TO  ONE  ONE  ONE  ONE  ONE  ONE  ONE 10      (SUBGOAL OF 24)
            SET: Y = TWO
            OBJECT 11: (LETTER B RELATION SAME INTERVAL TWO, LETTER C RELATION NEXT INTERVAL
                        TWO, LETTER B RELATION SAME INTERVAL TWO, LETTER D RELATION NEXT INTERVAL
```

Figure 85: (continued)

```
                TWO, LETTER 3 RELATION SAME INTERVAL TWO, LETTER E RELATION NEXT INTERVAL
                TWO, LETTER 3 RELATION LAST INTERVAL TWO, LETTER F RELATION LAST INTERVAL
                TWO)

          9  GOAL 27 TRANSFORM 11 INTO DESIRED-OBJ      (SUBGOAL OF 23)
SUCCESS
```

Figure 85: (continued)

a. LETTER B
b. CURRENT-RELATION SAME
c. INTERVAL TWO.

This indicates that the third position in OBJECT 4 corresponds to the current letter and that the previously assigned relationship is the current letter is the same as the second letter to the right of it.

In attempting GOAL 4, GPS notices that SAME is the CURRENT-RELATION of the third position. To remove this attribute-value pair (GOAL 5), APPLY-CURRENT-RELATION is applied to the third position in OBJECT 4 (GOAL 6). Note that ONE indicates one place to the right, e.g., 'ONE ONE 4' in GOAL 5 and GOAL 6 refers to two places to the right of the first position which is the third position. OBJECT 5 is produced by assigning the relationship--the third letter is the same as the fifth letter.

B is substituted for the first blank in order to achieve GOAL 9 and F is substituted for the second blank in achieveing GOAL 22. The result of GOAL 26 is a complete description of the series and the success of GOAL 27 indicates that TOP-GOAL is successful.

Discussion

The formulation of the task in Fig. 83 has several peculiarities. The definition of simplicity of a relationship is buried in the VAR-DOMAIN of APPLY-RELATION. Since the MOVE-OPERATOR-METHOD tries legitimate variable specifications in the order in which they turn up, they must turn up in the correct order if the simplest relationship is to be assigned first. Thus, if the order of the two TESTS in VAR-DOMAIN were changed, the simplest relationship would not necessarily be assigned by an application of APPLY-RELATION. For example, GOAL 3 would assign the relationship--the second letter is the NEXT after the first letter

The communication between operators seems rather strange. An application of APPLY-RELATION in some sense designates to which position GPS will attempt to apply the next operator and that it will be either APPLY-CURRENT-RELATION or APPLY-LAST. These peculiarities are both derived from making GPS find the simplest description of the series. DESIRED-OBJ indicates that any complete description of the series is sufficient, and the simplicity of the description results from the order in which operators are applied.

Another peculiarity is replacing the GOAL of filling in the blank with the GOAL of finding a complete description of the series. Since there must be some restrictions on how the series is completed (else any answer would be correct), the latter GOAL seems to be a reasonable way of stating the problem.

The purpose of posing this task to GPS is not basically to illustrate how the letter series completion task can be formalized. This has been done more elegantly and efficiently in the work already cited. However, the performance of GPS in Fig. 85 does illustrate how this task can be approached by searching for a suitable description of the series in a space of descriptions. In this respect, the formulation of this task is similar to the formulation of the binary choice task in Feldman et al [12] (discussed on page 36).

L. GENERALITY OF GPS'S METHODS

In Chapter II we discussed two variants of the approach to the construction of a general problem solver that focusses on the problem solving techniques: One involves the construction of many specialized, but highly efficient, problem solving techniques, e.g., a library of ALGOL subroutines. The other involves the construction of a few problem solving techniques of wide applicability, e.g., heuristic search. In this respect, the extent to which the methods of GPS are applicable across tasks is important.

Table 1 illustrates which methods and basic processes were used in solving the eleven tasks discussed in this chapter. We chose to digitize the processing at the method level for two reasons: 1) A detailed description of each method is given in Chapter III. 2) Each task uses some of the methods but not all of them; on the average each task uses about half of the methods. There are also disadvantages to this way of displaying the activity. Some of the methods employ much of the same code. For example, both the MOVE-OPERATOR-METHOD and the MATCH-DIFF-METHOD use a large subroutine which applies a TEST to an OBJECT-SCHEMA. In addition, considerable detail disappears at the method level. This may be important. For instance, not all of the subparts of a method may be used in a given task.

In Table 1 the methods are categorized according to the basic processes (described in Fig. 36) that they perform. There are other basic processes which GPS performs which are not discussed in this report because they were not a consideration for generalizing the representation. The basic process for selecting a member from a SET (the methods for achieving a SELECT GOAL) are independent of the repre-

Basic Processes	Methods \ Tasks	Missionaries & Cannibals	Integration	Tower of Hanoi	Predicate Calculus	Father & Sons	Monkey	Three Coins	Water Jug	Bridges of Konigsberg	Parsing	Letter Series
Object-Comparison, Object-Difference	TRANSFORM-METHOD	X	X	X	X	X	X	X	X	X	X	X
	MATCH-DIFF-METHOD	X	X	X	X	X	X	X	X	X	X	X
Operator-Application, Operator-Difference, Feasibility-Selection	MATCH-DIFF-METHOD		X		Y						X	
	MOVE-OPERATOR-METHOD	X		X		X	X	Y	X	X	X	Y
	FORM-OPERATOR-METHOD		X		Y						X	
	FORM-OPERATOR-TO-SET-METHOD											
	TWO-INPUT-OPERATOR-METHOD				Y							
	SET-OPERATOR-METHOD		X									
Desirability-Selection	REDUCE-METHOD	X	X	X	X	X	X	X	X	X	X	X
Canonization	IDENTITY-MATCH-METHOD	X	X	X	X	X	X	X	X	X	X	X
Other Processes	TRANSFORM-SET-METHOD	X			X	X			X			
	SELECT-BEST-MEMBERS-METHOD	X	X		X	X			X			
	GENERATE-AND-TEST-METHOD											
	EXPANDED-TRANSFORM-METHOD	X				X			X			
	ANTECEDENT-GOAL-METHOD	X			X	X			X	X	X	
	TRY-OLD-GOALS-METHOD								X			

X denotes that the method and basic process(es) were used in solving the task

Y denotes that the method and basic process(es), except for operator-difference were used in solving the task

TABLE 1. Methods and processes used by GPS in solving the eleven tasks.

sentation because the selection criterion is given to GPS as an IPL routine which may be as specialized as necessary. Other basic processes depend only on the GOAL structure of GPS. For example, GPS has processes for generating new things to do when it gets into trouble, such as generating an old GOAL.

Since the X's and Y's in Table 1 are scattered somewhat randomly, the methods do not appear to be task specialized. However, much of the code in GPS is specialized for a particular kind of representation. This can be seen in Table 1 by noting what methods for applying an operator are used by the various tasks. In fact, two methods--FORM-OPERATOR-TO-SET-METHOD and GENERATE-AND-TEST-METHOD--are not used by any of the tasks, because they are specialized to a particular aspect of the representation that does not arise in solving the tasks. (They were used for logic in GPS-2-5.) However, specialization by kind of representation is not necessary, as is demonstrated by the parsing task, which uses both the FORM-OPERATOR-METHOD and the MOVE-OPERATOR-METHOD.

All of the basic processes (described in Fig. 36), with the exception of operator-difference, are used in attempting each of the eleven tasks, which indicates that means-ends analysis is applicable to all of the tasks. Operator-difference does not occur in those tasks that were so simple that GPS finds a solution before attempting to apply an operator that is inapplicable. Similarly, the TRY-OLD-

GOALS-METHOD is used only in solving the water jug task because it is the only task for which GPS becomes hardpressed for something to do.

CHAPTER VII: SUMMARY

The presentation of the material is now complete. In Chapter II we formulated a problem of generality for problem solving programs, such that we could investigate it by getting GPS to perform a large number of different tasks. In Chapters III and IV we gave a detailed description of GPS, both its program structure and its representations of components of the task. In Chapter V we analysed the problems of extending GPS, revealing thereby a number of ways in which the representation is a critical aspect. Finally, in Chapter VI we described the several tasks we used, and drew a number of smaller lessons. We ended by observing that GPS was in fact a single problem solver working on a collection of tasks, and not simply a collection of mechanisms separately specialized for each task.

This presentation has involved a great deal of detail. Undoubtedly, we have told many readers more than they wish to know. Consequently, the purpose of this summary is to provide a brief, but complete, recapitulation of the total story.

A. THE APPROACH

Heuristic Search

This research approaches the construction of a general problem solver by first adopting a general paradigm of a problem, heuristic search (Newell and Ernst [38]). In a simplified form of the heuristic search paradigm, there are objects and operators, such that an operator can be applied to an object to produce either a new object or a signal which indicates inapplicability. A heuristic search problem is:

Given: a. an initial situation represented as an object,

b. a desired situation represented as an object,

c. a set of operators.

Find:

a sequence of operators that will transform the initial situation into the desired situation.

The first operator of the solution sequence is applied to the initial situation, the other operators are applied to the result of the application of the preceding operator, and the result of the application of the last operator in the sequence is the desired situation.

The operators are rules for generating objects and thus define a tree of objects. Each node of the tree represents an object, and each branch of a node represents the application of an operator to the object represented by the node. The node to which a branch leads represents the object produced by the application of the operator. A method for solving a heuristic search problem is searching the tree, defined by the initial situation and the operators, for a path to the desired situation.

For many problems we know of no obvious heuristic search formulation. Thus, in some sense adopting heuristic search limits the generality that can be achieved. However, heuristic search derives its appeal from its generality, demonstrated by its wide use in other research efforts into problem solving (discussed in Chapter II).

The Problem of Generality

The power of a problem solver is indicated by the effectiveness of its problem solving techniques while its generality is indicated by the domain of problems that it can deal with. The generality and the power of a problem solver are not independent because both depend strongly upon the internal representation. The internal representation is pulled in two directions: on the one hand, it must be general enough so that problems can be translated into it, and, on the other hand it must be specific enough so that the problem solving techniques can be applied.

To illustrate the interdependence of the power and the generality of a problem solver consider a heuristic search problem solver whose only technique is to generate objects by applying the operators in a fixed order and testing if any of the generated objects are identical to the desired situation. It would be easy to construct such a problem solver with a relatively high degree of generality even though it could only solve the most elementary problems. On the other hand, it would be difficult today to achieve even a slight degree of generality with a problem solver that discovered the terms in an evaluation function for determining the likelihood of the existence of a path from any object to the desired situation. Thus, there are many different problems of generality, one for each set of

problem solving techniques, and the difficulty of achieving generality depends upon the variety and complexity of the techniques.

This research investigates a particular problem of generality--the problem of extending the generality of GPS while holding its power at a fixed level. This involves extending the internal representation of GPS in such a way that its problem solving methods remain applicable and in a way that increases the domain of problems that can be translated into its internal representation. Thus, this research is mainly concerned with representational issues. We would not expect the issues to be the same in generalizing the internal representation of a problem solver which employed markedly different techniques than GPS. In this respect, this research has the nature of a case study.

B. GPS

GPS attempts problems by tree search, as in any heuristic search program. But to guide the search GPS employs a general technique called means-ends analysis which involves subdividing a problem into easier sub-problems. Means-ends analysis is accomplished by taking differences between what is given and what is wanted, e.g., between two objects or between an object and the class of objects to which an operator can be applied. A difference designates some feature of an object which is undesirable. GPS

uses the difference to select a desirable operator--one which is relevant to reducing the difference. For example, in attempting the original problem, GPS detects a difference, if one exists, between the initial situation and the desired situation. Assuming that a desirable operator exists and that it can be applied to the initial situation, GPS applies it and produces a new object. GPS rephrases the original problem by replacing the initial situation with the new object and then recycles. The problem is solved when an object is generated that is identical to the desired situation.

The problem solving techniques of GPS consist of a set of methods, which are applied by a problem solving executive. To solve a problem, the problem solving executive selects a relevant method and applies it. Subproblems (GOALs) may be generated by the method in an attempt to simplify the problem. In such cases the main problem may be temporarily abandoned by the problem solving executive for the purpose of solving the subproblem. Subproblems are attempted in the same way that the main problem is attempted--by selecting and applying a relevant method.

Chapter III gives a complete description of the problem solving executive and the methods. For the purpose of this chapter we need only summarize the demands that the problem solving methods of GPS place on the internal repre-

sentation. Each of these demands requires that GPS employ a process for abstracting certain information from the internal representation. These processes may be different for different representations, but the information abstracted does not depend on the representation.

Object-comparison. GPS must be able to compare two objects to determine if they represent the same situation.

Object-difference. If two objects do not represent the same situation, GPS must be able to detect differences between them that summarize their dissimilarity.

Operator-application. GPS must be able to apply an operator to an object. The result of this process is either an object or a signal that the application is not feasible.

Operator-difference. If it is infeasible to apply an operator to an object, GPS must be able to produce differences that summarize why the application is infeasible.

Desirability-selection. For any difference GPS must be able to select from all operators of a task those operators that are relevant to reducing the difference. (Of course, this selection will not in general be perfect.)

Feasibility-selection. For any object GPS must be able to select from all the operators those that will be applicable to the object. This is meant to cover the case where the

internal representation permits several operators of limited range to be combined in a single operator of wider range, such that the application of the unified operator does not decompose simply to the sequential application of the two suboperators. (Again, feasibility-selection need not be perfect.)

Canonization. GPS must be able to find the canonical name of certain types of data structures. Canonization arises from GPS's strategy for comparing two data structures. If they have canonical names, they are equivalent only if they have the same name. On the other hand, if two data structures do not have canonical names, they are equivalent only if all of their structure is equivalent.

C. INTERNAL REPRESENTATION OF GPS-2-5

The current version of GPS was developed through the modification of an existing version, called GPS-2-5. GPS-2-5, together with its predecessors, solved only three different kinds of problems. This was due mainly to inadequate facilities for representing tasks. The internal representation of GPS-2-5 will be described to clarify how the generalizations of the representation incorporated in GPS (described later) alleviated inadequacies in its representation.

The internal representation of a task for GPS (any version) consists of several different kinds of data structures:

a. objects
b. operators
c. differences
d. GOALs
e. TABLE-OF-CONNECTIONS
f. DIFF-ORDERING
g. details for matching objects
h. miscellaneous information.

Chapter IV gives a complete description of the above types of data structures and also points out which are given in the specification of a problem. Here, we will only describe the representation of objects and operators, which provide the main representational issues. Other than objects and operators, differences are the only other type of data structure whose representation in GPS is not the same as its representation in GPS-2-5. Their representation depends to a large extent on the representation of objects and operators. This will be discussed in more detail later.

Objects. In GPS-2-5 objects are represented by tree structures encoded in IPL description lists. Each node of the tree structure can have an arbitrary number of branches leading from it to other nodes. In addition to branches, each node can have a local description given by an arbitrary number of attribute-value pairs. The tree structure in Fig. 86

for example, represents the initial situation in the missionaries and cannibals task. In Fig. 86 the node to which the LEFT branch leads represents the left bank of the river and the node to which the RIGHT branch leads represents the right bank of the river. The local description at the node which the LEFT branch leads to indicates that there are three missionaries, three cannibals, and the boat at that bank of the river.

The use of variables in the tree structures described above allows a class of objects to be represented as a single data structure. For example, Fig. 87 is the tree structure representation of $\int e^u du$. If u is a variable, this tree structure represents a large class of objects. All members of the class have the same form but different values for u. GPS assumes that all tree structures may contain variables and it is prepared to process them as classes of objects.

Operators. In GPS-2-5 all operators were represented by representing the form of both the input and resultant objects. Assuming that u is a variable, Fig. 87 is the tree structure representation of the input of the operator, $\int e^u du = e^u$, and Fig. 88 is the tree structure of the output. Such an operator can only be applied to a member of the class of objects represented by the input form.

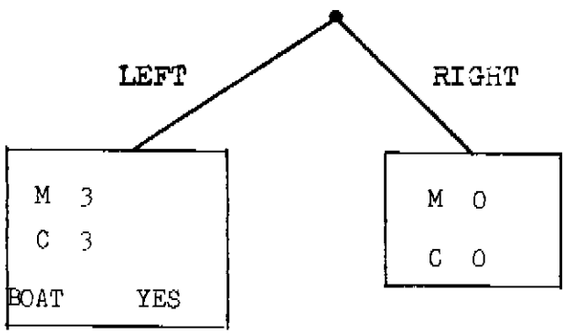


FIGURE 86. The tree structure representation of the initial situation in the missionaries and cannibal task.

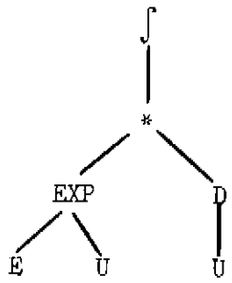


FIGURE 87. The tree structure representation of $\int e^{u}du$.

FIGURE 88. The tree structure representation of e^{u}.

D. REPRESENTATIONAL ISSUES

The representational issues that were investigated arose from various properties of tasks that could not adequately be dealt with by the existing program. Each of these issues will be discussed below. For some the representation was generalized, and the difficulty was removed. Other issues could not be dealt with within the framework of the existing program. However, attempting to alleviate these difficulties did clarify important aspects of the issues.

Desired Situation.

In many tasks the desired situation is a class of objects that could not be represented in GPS-2-5. In integration, for example, the desired situation is any expression that does not contain '$\int$'. A tree structure cannot represent this class of objects because all of the members do not have the same form. For this reason the representation of the desired situation had to be generalized.

In introducing a new representation for the desired situation, GPS must be given several new processes for abstracting information from the new representation: a new object-comparison process so that GPS can compare an object to the desired situation; and a new object-difference process so that GPS can detect differences between an object and the desired situation. Object-comparison and

object-difference are the only demands (listed on pages 320-1) of GPS's problem solving method that are affected by the introduction of a new representation for desired situation.

The generalization of the desired situation allowed it to be represented as a set of constraints called a DESCRIBED-OBJ. A set of constraints represents a class of objects, each of which satisfies all of the constraints. The desired object in the integration task can be represented by the single constraint:

No symbol in the expression is an '$\int$'.

Each constraint in a DESCRIBED-OBJ is a data structure, called a TEST, that consists of a RELATION, and several arguments (in most cases, two). In the previous example,

the RELATION is NOT-EQUAL;

the first argument is a symbol;

the second argument is '$\int$'.

This constraint is quantified "for all" symbols. GPS can recognize NOT-EQUAL as a RELATION which it understands. (Currently GPS understands fifteen RELATIONs.) On the other hand, GPS only understands the generic form of the arguments, which are task dependent.

Using constraints to represent objects is convenient because each constraint is a simple data structure. Both the object-comparison process and the object-difference process analyze the structure of the constraints. The structure

of many representations is too complex to permit such an analysis. For example, an alternate representation for the desired situation is a program whose input is an object and whose output is a signal indicating whether the object is a member of the class of objects that the program represents. The object-difference process for this representation would be extremely complex because it would require an analysis of the program.

Operators

The operators of many tasks, particularly mathematical calculi could be represented conveniently in GPS-2-5. However, the operators of other tasks could not, e.g., the operator of missionaries and cannibals. To alleviate this difficulty in GPS, an operator can be represented as a data structure, called a MOVE-OPERATOR, that consists of a group of TESTs and a group of TRANSFORMATIONs. The TESTs, which are the same as the TESTs in a DESCRIBED-OBJ, must be satisfied in order for the operator to be applicable and the TRANSFORMATIONs indicate how the resultant object differs from the input object.

A TRANSFORMATION is a data structure that consists of an OPERATION and several arguments. GPS knows the semantics of the OPERATIONs, but as in TESTs, only knows the generic form of the arguments, which are task dependent. Currently, GPS understands six OPERATIONs. A typical TRANSFORMATION

(from the missionaries and cannibals operator that moves X missionaries, Y cannibals and the BOAT from LEFT to RIGHT) is:

> DECREASE the number of missionaries at the LEFT by X and increase the number of missionaries at the RIGHT by X.

In this TRANSFORMATION the OPERATION is DECREASE and the arguments are X, the number of missionaries at the LEFT and the number of missionaries at the RIGHT.

Fig. 89 illustrates how the operator which moves X missionaries and Y cannibals from LEFT to RIGHT can be represented as a MOVE-OPERATOR. The first two TESTs indicate that X and Y must be greater than 0. The third TEST insures that at least one person is in the BOAT to operate it and that the capacity of the BOAT is not exceeded. The remaining TESTs prevent missionaries from being eaten.

The three TRANSFORMATIONs indicate how the application of the operator affects the number of missionaries, the number of cannibals and the BOAT, respectively. The BOAT must be at the LEFT in order for the third TRANSFORMATION to be applicable.

The introduction of MOVE-OPERATORs in GPS required the addition of new processes so that the problem solving methods could be applied to this new representation. New processes were needed operator-application, operator-difference, desirability-selection and feasibility-selection. Hence, the MOVE-OPERATOR representation was designed so as to make these

TESTs:

1. X $\{0, 1, 2\}$
2. Y $\{0, 1, 2\}$
3. $X + Y \leq 2$
4. Either
   a. the number of missionaries at the LEFT $\geq$ the number of cannibals at the LEFT,

   or

   b. the number of missionaries at the LEFT = 0.
5. Either
   a. the number of missionaries at the RIGHT $\geq$ the number of cannibals at the RIGHT,

   or

   b. the number of missionaries at the RIGHT = 0.

TRANSFORMATIONs:

1. DECREASE the number of missionaries at the LEFT by X and increase the number of missionaries at the RIGHT by X.
2. DECREASE the number of cannibals at the LEFT by Y and increase the number of cannibals at the RIGHT by Y.
3. MOVE the BOAT from the LEFT to the RIGHT.

FIGURE 89. The MOVE-OPERATOR representation of the operator that moves X missionaries, Y cannibals and the BOAT from the LEFT to the RIGHT.

processes simple. For many representations one or more of these processes would be too complex to implement.

A key feature of the MOVE-OPERATOR representation is its transparent structure. Each of the new processes does an analysis of this structure in order to abstract the necessary information. Another good property of MOVE-OPERATORs is its structural similarity to DESCRIBED-OBJ. This similarity causes the MOVE-OPERATOR processes to be similar to the DESCRIBED-OBJ processes, and thus all of these processes use the same basic subroutines. For example, the operator-difference process for MOVE-OPERATORs and the object-difference process for DESCRIBED-OBJ are nearly identical.

Unordered Sets

The representation of some tasks requires representation of an unordered set. Multiplication, for example, can be represented as an n-ary function of a set of arguments whose order is unimportant. Such an unordered set can be represented in GPS-2-5 as an object, representing an ordered set, and an operator for permuting the elements of the set. This representation has the drawback that discovering the identity of two sets requires several applications of the permutation operators. The permutation operator would be unnecessary if the identity test could implicitly take into consideration the unordered property of the two sets.

The objects of GPS-2-5 can be used to implicitly represent unordered sets, provided that the nodes can be tagged either ordered or unordered. These tags designate the branches of a node to be either ordered or unordered. Although a seemingly simple generalization, it considerably complicates the object-comparison, object-difference, the operator-application, the operator-difference, and the canonization process. These processes were generalized for the integration task so that the nodes of objects and operators could be unordered. Although the generalized processes were more complex and did more processing, there was a savings due to an overall reduction in the problem space.

The main feature of unordered sets that complicates the processes is the fact that in matching two unordered sets corresponding elements must be paired. A variable can be made identical to any element via substitution and thus can be paired with any element. However, to see the identity of two unordered sets may require that a particular variable be paired with a particular element. Chapter V discusses this issue in more detail and describes how the generalized processes (object-comparison, etc.) deal with this issue.

Large Objects

GPS can only solve simple problems before its memory is

exhausted. However, for some tasks the objects are so large that not even simple problems can be solved before its memory is exhausted. For example, the representation of a chess board in GPS requires 1,000 memory locations and thus only several objects can be stored in memory.

There are two distinct difficulties with GPS's use of memory: 1) GPS saves in memory all objects generated during problem solving and 2) each object is a total situation, i.e., there is no provision for dealing with fragments of situations. These difficulties could not be dealt with in this research because they are too closely connected with the problem solving methods of GPS, which were held fixed.

Differences

In generalizing GPS, the representation of differences was degenerated. Each difference in GPS can only pertain to the value of an attribute of a node of an object. More global differences, such as the number of occurances of a symbol which could be represented in GPS-2-5, cannot be represented because they would introduce considerable complexity in the operator-difference, the object-difference, and the desirability-selection processes. Thus, the generalization of these processes for MOVE-OPERATORs and DESCRIBED-OBJs was based on this simplified representation of differences.

Differences, although not part of the general heuristic search paradigm, are central to means-ends analysis, which is the main technique of GPS.

Many tasks were not given to GPS, because the simple differences would not adequately guide GPS's search for a solution. For example, many of the logic tasks solved by GPS-2-5 cannot be solved by GPS due to the lack of direction provided by the degenerate differences. However, the representation of differences is adequate for the eleven tasks that were given to GPS.

E. TASKS

One of the instructive aspects of this research is the light shed upon the structure of the problems given to GPS. It is for this reason that GPS was given rather elaborate input-output facilities. It is hoped that the reader, if he desires, can decipher the input and output for any of the eleven tasks discussed in Chapter VI after reading a description of the input and output. If this is the case, the reader can determine precisely what information is given to GPS in the task specification and see how GPS uses this information in solving the task.

Each task is discussed in detail in Chapter VI. Here we will only summarize the important aspects of each.

Missionaries and Cannibals

GPS and one of its predecessors, called GPS-2-2, both

solved the missionaries and cannibals task. The representation of the task in GPS was quite different from that used by GPS-2-2. The latter contains information about the nature of operators which the current GPS discovers for itself. GPS-2-2 was given ten operators: Move one missionary from left to right; move two missionaries from left to right: move one missionary and one cannibal from left to right, etc. The desirability of these operators for reducing the various types of differences was given to GPS-2-2, exogenously. GPS is only given a single operator which moves X missionaries and Y cannibals across the river. In applying this operator GPS specifies the variables (X, Y, and the direction of the boat) so that the operator performs a desirable function.

GPS-2-2 was given a desirability filter for operators. This filter prevented GPS-2-2 from attempting to move more missionaries and cannibals across the river than there were on the side from which they were being moved. Such a separate filter is unnecessary in GPS because GPS never considers applying such an operator. Each operator in the GPS-2-2 formulation consisted of an IPL routine with its parameters (described on pages 30-3). The operator filter was also encoded in IPL. Not only is it tedious to construct IPL routines but the construction of these routines required some knowledge of the internal structure of GPS-2-2. The construction of the single operators given to GPS is much less

tedious and requires no knowledge of the internal structure of GPS.

Integration

In the integration, multiplication and addition are represented as n-ary functions whose arguments are represented as an unordered set. Thus the commutativity and associativity of multiplication and addition are expressed implicitly instead of representing them explicitly as operators. If they were explicitly given to GPS as operators, there would be an overall increase in the problem space which would prevent GPS from solving some trivial integrals.

SAINT [59], a program that is quite proficient at symbolic integration, also represents the commutativity and associativity of multiplication and addition implicitly. Other similarities and some dissimilarities between SAINT and GPS are discussed in Chapter VI.

Tower of Hanoi

The Tower of Hanoi is an example of a task for which means-end analysis is very effective. GPS never makes a mistake on this task, mainly because the differences and the DIFF-ORDERING are in some sense optimal. For many tasks, it is difficult to find good differences and a good DIFF-ORDERING.

Proving Theorem in the Predicate Calculus

GPS proved a simple theorem in the first order predicate calculus. The formulation of this problem is basically the

same as the one in Robinson [51]. Perhaps the most instructive part of this example is the light it cast upon the evolution of problem solving programs. In LT (Newell, et al [39]), a theorem proving program for the propositional calculus, which is the predecessor of GPS, it was noted that the match routine was the source of most of the power of the program over a brute force search. GPS may be considered as an attempt to generalize the match routine, based on that experience. The first predicate calculus theorem prover did in fact use brute force search, Gilmore [17]. From an efficiency point of view the main effect of the resolution principle was to reintroduce the possibility of matching (gaining, thereby, a vast increase in power). And it is this feature that allows GPS to use the resolution principle in a natural way.

Father and Sons

This task is very similar to the missionaries and cannibals task. Both tasks involve moving two different kinds of people across a river in a small boat. But their formulations for GPS are quite different, in that none of the operators, objects, or differences are the same. Many of the earlier publications on GPS (e.g., Newell, et al, [42] and Newell, et al [45]) make the distinction between a task and a task environment -- the common part of a group of similar tasks. The father and sons task and the missionaries and cannibals task muddles this

distinction. On the one hand, they should both have the same task environment because of their similarity. In fact they have different task environments because none of their objects, operators and differences are the same.

Monkey Task

The monkey task was created by McCarthy [27] as a typical problem for the Advice Taker [26]. It is interesting to compare GPS's formulation of this task to the formulation for a typical Advice Taker program (Black [4]). In GPS the objects are models of room configurations whereas the objects in Black [4] are linguistic expressions that describe certain aspects of the room configurations. Both representations have advantages. For example, linguistic expressions are useful for representing imperfect information such as the monkey is in one of two places. However, models can represent implicit information that has to be represented explicitly when linguistic expressions are used, e.g., the monkey can only be in one place at a time.

Three Coins

The peculiarity of the three coins task is in the solution being constrained to a fixed number of operator applications. This constraint was handled in GPS by expanding the representation of objects to include a counter that indicates the number of operator applications involved in producing the objects. In the desired situation the counter must have a

particular value.

Parsing Sentences

A great deal of effort has been devoted to the construction of efficient parsing algorithms for simple phase structure grammars. The point of this example is not GPS's proficiency as a parser, but to illustrate the kinship between heuristic search and syntactic analysis.

Bridges of Königsberg

This is the only impossible task that was given to GPS. Although GPS's behavior is not aimless (it crosses six bridges in two different ways), GPS cannot see the impossibility because it lies in the topological properties of the bridges. GPS only attempts to cross bridges and has no way of viewing the problem as a whole.

Water Jug

For the water jug task, means-ends analysis seems to be a rather ineffective heuristic as demonstrated by the fact that GPS stumbled onto the solution. There are some better differences although complex (involving modular arithmetic, naturally).

Letter Series Completion

The letter series completion task is the only task whose solution requires inductive reasoning. The formulation is quite clumsy, but this example demonstrates how the problem can be approached by searching for a suitable description in a space of descriptions. The binary choice task, another

task requiring inductive reasoning, was formulated in a similar way by Feldman, et al [12].

With these eleven tasks in hand one can examine the extent to which GPS uses common mechanisms to solve its problems, rather than (in essence) consisting of a big switch to separate subroutines that are specialized to each task. A tabulation of methods versus tasks (Table I, page 312) indicate that GPS is an integrated system, each method being used in an average of half the tasks. The comparison is not perfect, however, since on the one hand the methods have common subparts, and on the other the use of a method does not indicate use of all its subparts.

BIBLIOGRAPHY

1. Amarel, S., "On the Automatic Formation of a Computer Program which Represents a Theory," Self-Organizing Systems, Yovits, M.C., Jacobi, G., and Goldstein, G. (eds.), Spartan Books, Washington, D.C., 1962.

2. Baylor, G. W., Report on a Mating Combinations Program, Master's Thesis, Department of Psychology, Carnegie Institute of Technology, Pittsburgh, Pa., 1965.

3. Bernstein, A., et al., "A Chess-Playing Program for the IBM 704," Proceedings of the 1958 Western Joint Computer Conference, (May 1958), 157-159.

4. Black, Fischer, A Deductive Question Answering System, Doctoral Dissertation, Division of Engineering and Applied Physics, Harvard University, Cambridge, Mass., 1964.

5. Bobrow, D. G., Natural Language Input for a Computer Problem Solving System, Doctoral Dissertation, Mathematics Dept., Massachusetts Institute of Technology, Cambridge, Mass., 1964.

6. Chomsky, A. N., Syntactic Structures, Mouton and Co., The Hague, The Netherlands, 1957.

7. Church, A., Introduction to Mathematical Logic, Princeton University Press, Princeton, N. J., 1956.

8. Davis, M., Computability and Unsolvability, McGraw-Hill, New York, 1958.

9. Davis, M., and Putnam, H., "A Computing Procedure for Qualification," J. ACM, 7, (July 1960), 201-215.

10. Feigenbaum, E., "An Information Processing Theory of Verbal Learning," Doctoral Dissertation, Graduate School of Industrial Administration, Carnegie Institute of Technology, Pittsburgh, Pa., 1959.

11. Feigenbaum, E. A. and Feldman, J. (eds.), Computers and Thought, McGraw-Hill, New York, 1963.

12. Feldman, J., Tonge, F. and Kanter, H., "Emperical Explorations of a Hypothesis-Testing Model of Binary Choice Behavior," Symposium on Simulation Models, Hoggott, A. C. and Balderston, F. E. (eds.) South-Western Publishing Co., Cincinnati, Ohio, 1963, 55-100.

13. Filipiak, A.S., 100 Puzzles: How to Make and How to Solve Them, A. S. Barnes & Co., New York, 1942, 20-21.

14. Friedman, J., "A Semi-decision Procedure for the Functional Calculus," J. ACM, 10, (Jan. 1963), 1-24.

15. Gelernter, H., "Realization of a Geometry-Theorem Proving Machine," Proceedings of an International Conference on Information Processing, UNESCO House, Paris, 1959, 273-282; reprinted in Feigenbaum and Feldman [11].

16. Gilbert, W. L., Private communication.

17. Gilmore, P. C., "A Proof Method for Quantification Theory," IBM J. Res. Develop., 4, (Jan. 1960), 28-35.

18. Green, B. B., Wolf, A. K., Chomsky, C., and Laughery, K.,"BASEBALL: An Automatic Question Answerer," Proceedings of the Western Joint Computer Conference, 1961, 219-224; reprinted in Feigenbaum and Feldman [11].

19. Hormann, Aiko, "Gaku: An Artificial Student," Behav. Sci., 10, (Jan. 1965), 88-107.

20. Katona, G., Organizing and Memorizing: Studies in the Psychology of Learning and Teaching, Columbia University Press, New York, 1940.

21. Kister, J., Stein, P., Ulam, S., Walden, W. and Wells, M., "Experiments in Chess," J. ACM, 4, (April 1957), 174-177.

22. Kotok, A., A Chess Playing Program for the IBM 7090, Bachelor's Thesis, Massachusetts Institute of Technology, Cambridge, Mass., 1962.

23. Krulee, G. K., and Kuck, D. J., "A Problem Solver with Formal Descriptive Inputs," Computer and Information Sciences, Tou, J. T. and Wilcox, R. H. (eds.), Spartan Books, Washington, D. C., 1964, 344-374.

24. Lindsay, R. K., "Inferential Memory as the Basis of Machines which Understand Natural Language," Computers and Thought, Feigenbaum, E. A. and Feldman, J. (eds.), McGraw-Hill, New York, 1963, 219-233.

25. Luchins, A. S. and Luchins, E. H., Rigidity and Behavior: A Variational Approach to the Effect of Einstellung, University of Oregon Books, Eugene, Oregon, 1959.

26. McCarthy, J., "Programs with Common Sense," Proc. Symposium on Mech. of Thought Processes, Her Majesty's Stationery Office, London, 1959, 75-84.

27. McCarthy, J., Situations, Actions, and Causal Laws, Stanford Artificial Intelligence Project Memo No. 2, July, 1963.

28. McCarthy, J., Private communication.

29. McCarthy, J., et al., LISP 1.5 Programmers Manual, MIT Press, Cambridge, Mass., 1963.

30. Moore, O. K. and Anderson, S. B., "Modern Logic and Tasks for Experiments on Problem-Solving," Journal of Psychology, 38, 1954, 151-160.

31. Morehead, A. H., and Mott-Smith, G., Hoyle's Rules of Games, New American Library of World Literature, New York, 1963.

32. Mott-Smith, G., Mathematical Puzzles for Beginners and Enthusiasts, Dover Publications, 1954.

33. Newell, A. (ed.), Information Processing Language-V Manual, second edition, Prentice Hall, Englewood Cliffs, N. J., 1961.

34. Newell, A., "Some Problems of Basic Organization in Problem-Solving Programs," Self-Organizing Systems, Yovits, M. C., Jacobi, G. T., and Goldstein, G. D. (eds.), Spartan Books, Washington, D. C., 1962, 393-425.

35. Newell, A., A Guide to the General Problem-solver Program GPS-2-2, RAND Corporation, Santa Monica, Calif., RM-3337-PR, 1963.

36. Newell, A., "Learning, Generality and Problem-Solving," Proceedings of IFIP Congress 62, North Holland Publishing, Amsterdam, Holland, 1962.

37. Newell, A., "Limitations of the Current Stock of Ideas about Problem Solving," Electronic Information Handling, Kent, A. and Taulbee, U. (eds.), Spartan Books, Washington, D. C., 1965.

38. Newell, A., and Ernst, G., "The Search for Generality," Proc. of IFIP Congress 65, Kalenich, W. A. (ed.), Spartan Books, Washington, D. C., 1965, 17-24.

39. Newell, A., Shaw, J. C., and Simon, H., "Empirical Explorations with the Logic Theory Machine," Proceedings of the Western Joint Computer Conference, 1957, 218-239; reprinted in Feigenbaum and Feldman [11].

40. Newell, A., Shaw, J. C. and Simon, H. A., "Preliminary Description of General Problem-Solving Program--I (GPS-1)," CIP Working Paper No. 7, Graduate School of Industrial Administration, Carnegie Institute of Technology, Pittsburgh, Pa., Dec. 1957.

41. Newell, A., Shaw, J. C., and Simon, H., "Chess Playing Programs and the Problem of Complexity," IBM Journal of Research and Development, October, 1958, 320-335; reprinted in Feigenbaum and Feldman [11].

42. Newell, A., Shaw, J. C., and Simon, H. A., "Report on a General Problem-Solving Program for a Computer," Information Processing: Proceedings of the International Conference on Information Processing, 256-264, UNESCO, Paris, 1960; reprinted in Computers and Automation, July, 1959.

43. Newell, A., and Simon, H. A., "Computer Simulation of Human Thought," Science, 134, December, 1961, 2011-2017.

44. Newell, A., and Simon, H., "GPS, a Program that Simulates Human Thought," Lernende Automaten, Munich, Germany, 1961; reprinted in Feigenbaum and Feldman [11].

45. Newell, A., Simon, H. A., and Shaw, J. C., "A Variety of Intelligent Learning in a General Problem-Solver," Self-Organizing Systems, Yovits, M. C., and Cameron S. (eds.), Pergamon Press, New York, 1960, 153-189.

46. Northrop, E. P., Riddles in Mathematics: A Book of Paradoxes, D. VanNostrand Co., New York, 1944, 65-66.

47. Oettinger, A. G., "Automatic Processing of Natural and Formal Languages," Proc. of IFIP Congress 65, Kalenich, W. A. (ed.), Spartan Books, Washington, D. C., 1965, 9-16.

48. Pivar, M. and Finkelstein, M., "Automation, Using LISP, of Inductive Inference on Sequences," The Programming Language LISP: Its Operation and Applications, Berkeley, E. and Bobrow, D. (eds.), Information International, Cambridge, Mass., 1964.

49. Raphael, B., SIR: A Computer Program for Semantic Information Retrieval, Doctoral Dissertation, Mathematics Dept., Massachusetts Institute of Technology, Cambridge, Mass., 1964.

50. Robinson, J. A., "Theorem-proving on the Computer," J. ACM, 10, (Apr. 1963), 163-174.

51. Robinson, J. A., "A Machine-Oriented Logic Based on the Resolution Principle," J. ACM, 12, (Jan. 1965), 23-41.

52. Rosenblatt, F., Principles of Neurodynamics, Spartan Books, Washington, D. C., 1962.

53. Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers," IBM Journal of Research and Development, 3, (July 1959), 211-229; reprinted in Feigenbaum and Feldman [11].

54. Simon, H. A., "Experiments with a Heuristic Compiler," J. ACM, 10, (Oct. 1963), 493-506.

55. Simon, H. A. and Kotovsky, K., "Human Acquisition of Concepts for Sequential Patterns," Psychological Review, 70, (June 1963), 534-546.

56. Simon, H. A. and Newell, A., "The Simulation of Human Thought," Current Trends in Psychological Theory, University of Pittsburgh Press, Pittsburgh, Pa., 1961, 152-179.

57. Simon, H. A. and Newell, A., "Computer Simulation of Human Thinking and Problem Solving," Management and the Computer of the Future, Greenberger, M. (ed.), John Wiley and Sons, New York, 1962; Computers and Automation, April, 1961, 18-19, 22-24, 26; Datamation, June, 1961, 18-20; and July, 1961, 35-37.

58. Simon, H. A., Newell, A., and Shaw, J., "The Processes of Creative Thinking, Contemporary Approaches to Creative Thinking, Gruber, H. E., Terell, G. and Wertheimer, M. (eds.), Atherton Press, New York, 1962, 63-119.

59. Slagle, J. R., "A Heuristic Program that Solves Symbolic Integration Problem in Freshman Calculus," J. ACM, 10, (Oct. 1963), 335-337; reprinted in Feigenbaum and Feldman [11].

60. Slagle, J. R., "Experiments with a Deductive Question-Answering Program," Comm. ACM, 8, (Dec. 1965), 792-798.

61. Tonge, F., "Balancing Assembly Lines Using the General Problem Solver," Symposium on Simulation Models, Hoggott, A. C., and Balderston, F. E. (eds.), Southwestern Publishing Co., Cincinnati, Ohio, 1963, 139-151.

62. Wang, Hao, "Proving Theorems by Pattern Recognition, I," Comm. ACM. 3, (1960), 220-234.

63. Wang, Hao, "Proving Theorems by Pattern Recognition, II," Bell System Tech. J., 40, (1961), 1-41.

64. Weizenbaum, J., "How to Make A Computer Appear Intelligent," Datamation, (February 1962), 24-26.

65. Williams, D., Unpublished research.

66. Williams, T. G., Some Studies in Game Playing with a Digital Computer, Doctoral Dissertation, Department of Electrical Engineering, Carnegie Institute of Technology, Pittsburgh, Pa., 1965.

67. Wos, L., Carson, D., and Robinson, G., "The Unit Preference Startegy in Theorem Proving," AFIPS Conference Proceedings 26, Spartan Books, Washington, D. C., 1964, 615-621.

APPENDIX A: THE VOCABULARY OF GPS

1. Meta-words

The following words are instructions to the process that translates the external representation into the internal representation on how to interpret the text:

DECLARE -- sets the mode of the translator to DECLARE so that it will designate words to be a particular type of symbol or data structure.

END -- signifies the end of a task description.

LIST -- sets the mode of translation to LIST so that the words following it are translated as data structures consisting of strings of symbols to be converted into the internal representation after translation.

RENAME -- sets the mode of the translator to RENAME so that it can assign new names to the words in the basic vocabulary of GPS.

SKIP-WORDS -- sets the mode of the translator to SKIP-WORDS so that it will designate the words following it to be ignored.

TASK-STRUCTURES -- sets the mode of translator so that it will interpret the words following it as data structures.

, -- in the LIST mode of translation is a text-word. In the other modes, it marks the end of a group of words.

. -- marks the end of a group of words.

(-- marks the beginning of a group of words.

) -- marks the end of a group of words.

2. Text words

The following words comprise that part of the basic vocabulary of GPS used in the representation of tasks. GPS understands each of these words, i.e., each corresponds to an IPL symbol which appears in some of the processes of GPS.

AMOUNT -- indicates that following it is the third argument, required by some OPERATIONs, of a TRANSFORMATION.

APPLY -- is the type of the GOAL of applying an operator to an object.

ATTRIBUTE -- is the type assigned to words which are attributes of nodes of OBJECT-SCHEMAs.

COMMON-DIFFERENCE -- stands for all types of differences.

COMPARE-OBJECTS -- is a parameter to the process that matches two objects.

CONSTRAINED-MEMBER -- is a RELATION on three arguments. It requires that its third argument, which must be a TEST, is true and that its first argument is in the set designated by its second argument.

CONSTRAINT -- precedes the thirdargument of a TEST whose RELATION is CONSTRAINED-MEMBER.

COPY -- is the OPERATION that places a copy of its first argument at the FEATURE which is its second argument.

CREATION-OPERATOR -- is the kind of MOVE-OPERATOR that creates new resultant objects.

DECREASE -- is the OPERATION that decrements its first argument by its third argument and increments its second argument by its third argument. The first and second arguments must be FEATUREs, and the third argument follows AMOUNT.

DEFINED -- is the RELATION that requires its first argument to have a value other than UNDEFINED. It only has one argument.

DESCRIBED-OBJ -- is the type of object which is represented by a group of TESTs.

DIFF-ORDERING -- orders the type of difference according to their relative difficulty.

EQUALS -- is the RELATION that requires its two arguments to designate the same value.

EXCLUSIVE-MEMBER -- is the RELATION that requires its first argument to be a unique member of the SET designated by its second argument.

EXPRES -- is the type of data structure that is an arithmetic expression.

FEATURE -- is the type of data structure that designates a feature of an object.

FIRST -- is the LOC-PROG that designates the first subnode of the implied node of an OBJECT-SCHEMA.

FIRST-FIRST -- is the LOC-PROG that designates the FIRST node of the FIRST node of the implied node of an OBJECT-SCHEMA.

FIRST-FIRST-FIRST -- is the LOC-PROG that designates the FIRST node of the FIRST node of the FIRST node of the implied node of an OBJECT-SCHEMA.

FIRST-SECOND -- is the LOC-PROG that designates the SECOND node of the FIRST node of the implied node of an OBJECT-SCHEMA.

FOR-ALL -- signifies that the TEST in which it occurs contains a universally quantified variable. The quantified variable follows FOR-ALL and the SET of values of the variables follows the variable.

FORM-OPERATOR -- is the type of operator that consists of an input form, which is an OBJECT-SCHEMA, and an output form, which is an OBJECT-SCHEMA.

FUNCTION -- indicates that following it is the name of a function.

GOAL -- is the type of data structure that represents a desired state of affairs and a history of previous attempts to achieve the GOAL.

GREATER-THAN -- is the RELATION that requires its first argument to be greater than its second argument.

INCREASE -- is the OPERATION that increases the value of its second argument by the amount of its first argument.

IN-THE-SET -- is the RELATION that requires its first argument to be in the SET designated by its second argument.

LESS-THAN -- is the RELATION that requires its first argument to be less than its second argument.

LIST-OF-OPR -- is a list of the FORM-OPERATORs in the task specification.

LIST-OF-VAR -- is a list of the variables in the task specification.

LOC-PROG -- is the type of the symbols that are the relative names of the nodes of OBJECT-SCHEMAs.

MOVE -- is the OPERATION that removes the value of its first argument and makes it the value of its second argument. Both arguments must be FEATUREs.

MOVE-FUNCTION -- is the OPERATION that removes the value of its first argument and makes the value of its second argument equal to a function of its first argument. The function follows FUNCTION.

MOVE-OPERATOR -- is the type of operator which consists of several groups of RELATIONs and a group of TRANSFORMATIONs.

MOVES -- signifies that a group of TRANSFORMATIONs will follow.

N-ARY-CONNECTIVE -- is a type of symbol. Any node (of an OBJECT-SCHEMA) that has this type of symbol as the value of an ATTRIBUTE has more than one sub-node. This information is used by the routines that convert objects and operators into their internal representation.

NOT-A-CONSTRAINED-MEMBER -- is the RELATION that is the negation of CONSTRAINED-MEMBER.

NOT-AN-EXCLUSIVE-MEMBER -- is the RELATION that is the negation of EXCLUSIVE-MEMBER.

NOT-EQUAL -- is the RELATION that is the negation of EQUALS.

NOT-GREATER-THAN -- is the RELATION that is the negation of GREATER-THAN.

NOT-IN-THE-SET -- is the RELATION that is the negation of IN-THE-SET.

NOT-LESS-THAN -- is the RELATION that is the negation of LESS-THAN.

OBJ-ATTRIBUTE -- is a list of all the words that are ATTRIBUTEs.

OBJECT-SCHEMA -- is the type of an object that is represented as a tree structure. OBJECT-SCHEMAs can contain variables in which case they represent a class of objects.

OPERATION -- is the type of symbol that designates the function of a TRANSFORMATION.

PARTICULAR -- precedes the name of the node of which the FEATURE (in which it occurs) is a function. PARTICULAR always occurs within the scope of a FEATURE which is not a function of the implied node.

POST-TESTS -- signifies that the group of TESTs that follow must be satisfied by the resultant object. This word only occurs in a MOVE-OPERATOR.

PRETESTS -- signifies that following it is the group of TESTs that represents the class of objects to which the operator can be applied. This word only occurs in a MOVE-OPERATOR.

QUOTE -- indicates that the word following it stands for itself.

REDUCE -- is the type of the GOAL of reducing a difference on an object.

REMOVE -- is the OPERATION that deletes the value of its first argument. It only has one argument.

RELATION -- is the type of symbol that designates the function of a TEST.

SELECT -- is the type of the GOAL of selecting an element of a SET.

SET -- is the type of data structure that is a set of items.

SET-SIZE -- is a type of difference that is produced by the commutive match used in the integration task. SET-SIZE is not a FEATURE.

SECOND -- is the LOC-PROG that designates the second subnode of the implied node of an OBJECT-SCHEMA.

SECOND-FIRST -- is the LOC-PROG that designates the FIRST node of the SECOND node of the implied node of an OBJECT-SCHEMA.

SECOND-SECOND -- is the LOC-PROG that designates the SECOND node of the SECOND node of the implied node of an OBJECT-SCHEMA.

SIGN -- is an ATTRIBUTE of the integration task and the predicate calculus task. The routine that converts objects and operators into their internal representation knows the meaning of SIGN.

SUBEXPRESSIONS -- designates that the match should detect differences at all nodes of OBJECT-SCHEMA. It only occurs in a COMPARE-OBJ.

SUBEXPRESSION-TESTS -- signifies that the set of TESTs that follows it must be true of every node of an OBJECT-SCHEMA. It only occurs in a DESCRIBED-OBJ.

SYMBOL -- is an ATTRIBUTE of the integration task and the predicate calculus task. The routine that converts objects and operators into their internal representation knows the meaning of SYMBOL.

TABLE-OF-CONNECTIONS -- associates with each type of difference the desirable operators.

TEST -- is a data structure consisting of a RELATION and its arguments.

THIRD -- is the LOC-PROG that designates the third subnode of the implied node of an OBJECT-SCHEMA.

TOP-GOAL -- is a statement of the problem.

TOP-NODE -- the LOC-PROG that designates the topmost node of an OBJECT-SCHEMA.

TRANSFORM -- the type of the GOAL of transforming one object into another.

TRANSFORMATION -- is a data structure consisting of an OPERATION and its arguments.

TRUE -- is a RELATION on a single argument. A TEST whose RELATION is TRUE is satisfied if one or more of the TESTs in the argument, which must be a set of TESTs, is satisfied.

UNARY-CONNECTIVE -- is a type of symbol. Any node (of an OBJECT-SCHEMA) that has this type of symbol as the value of an ATTRIBUTE has one subnode. This information is used by the routines that convert objects and operators into their internal representation.

UNDEFINED -- is the RELATION that requires that its first argument has the value, UNDEFINED. It only has one argument. UNDEFINED can also be the value of a FEATURE, an ATTRIBUTE, or a LOC-PROG.

VAR-DOMAIN -- indicates that following it is a group of TESTs that must be satisfied in order for the variables to have legitimate values. It only occurs in MOVE-OPERATORs.

V-TESTS -- is the type of a group of TESTs that follows TRUE.

YIELDS -- separates the input form and the output form of a FORM-OPERATOR.

+ -- is an arithmetic operation that appears in EXPRESs.

, -- in the LIST mode of translation is the text-word that separates the OBJECT-SCHEMAs in a SET of OBJECT-SCHEMAs and separates the input forms in a FORM-OPERATOR that has two objects as an input. In the other modes of translation, it is processed as a meta-word.

- -- is the value of the ATTRIBUTE, SIGN. The routine that converts the operators and objects of certain tasks into their internal representation knows this.

3. Skip-words

The following words are ignored by the translator unless otherwise specified:

A,ALL, AN, ANY, AT,AND, ADD, ARE, BE, DO, DOES, FOR, FROM, IS, IT, IN, INTO, NOT, ONE, ONES, ON, OR, OF, SHOULD, THAN, THE, -----, 1. , 2. , 3. , 4. , 5. , 6. , 7. , 8. , 9. , 10., BY, THAT.

APPENDIX B: THE OPERATORS OF THE LOGIC TASK

This appendix contains the specification of the one-input operators in the formulation of logic in Fig. 8 : They are expressed as FORM-OPERATORs in the first section and as MOVE-OPERATORs in the second section. The two input operators of the logic task are not included in this appendix because there is no provision for expressing two input operators as MOVE-OPERATORs. The names used for the operators in Fig. 8 are also used in this appendix. Some of the operators in Fig. 8 can only be expressed as several FORM-OPERATORs, each of which is assigned a local name, e.g., a, b,... In section 2, the MOVE-OPERATORs are assigned the same names. For example, the MOVE-OPERATOR R1: a,b is equivalent to the two FORM--OPERATORs, R1: a and R1: b.

1. FORM-OPERATORs

The logical connectives, conjunction, disjunction, implication, and negation, are symbolized as ., V, I, and -, respectively. A, B, C and X are free variables that stand for propositions.

R1: a. (A V B) YIELDS (B V A)
b. (A . B) YIELDS (B . A)

R2: (A I B) YIELDS (- B I - A)

R3: a. (A V A) YIELDS A
b. (A . A) YIELDS A

R4: a. (A V (B V C)) YIELDS ((A V B) V C)
b. (A . (B . C)) YIELDS ((A . B) . C)
c. ((A V B) V C) YIELDS (A V (B V C))
d. ((A . B) . C) YIELDS (A . (B . C))

R5: a. (A V B) YIELDS - (- A . - B)
b. - (- A . - B) YIELDS (A V B)

R6: a. (A I B) YIELDS (- A V B)
b. (- A V B) YIELDS (A I B)

R7: a. (A V (B . C)) YIELDS ((A V B) . (A V C))

b. (A . (B V C)) YIELDS ((A . B) V (A . C))

c. ((A V B) . (A V C)) YIELDS (A V (B . C))

d. ((A . B) V (A . C)) YIELDS (A . (B V C))

R8: a. (A . B) YIELDS A

b. (A . B) YIELDS B

R9: A YIELDS (A V X)

2. MOVE-OPERATORs

The notation used in the MOVE-OPERATORs is the same as above with the exception that PERIOD is used for conjunction because "." is used as punctuation in the specification of MOVE-OPERATORs. Additional information must be given for specifying the logic operators as MOVE-OPERATORs:

a. LEFT and RIGHT are used as the names of the LOC-PROG, FIRST, and SECOND.

b. SYMBOL and SIGN are the ATTRIBUTEs of a node of an OBJECT-SCHEMA.

c. <V,.> is the set of two elements, V and ..

d. <V,I> is the set of two elements, V and I.

e. [-+,+-] is the FUNCTION whose value is + if the input is - and - if the input is +.

f. [VI,IV] is the FUNCTION whose value is I if the input is V and V if the input is I.

g. [V.,.V] is the FUNCTION whose value is . if the input is V and V if the input is ..

The MOVE-OPERATOR representation of R1-R9 follows:

R1: a,b. (PRETESTS

THE SYMBOL IS IN-THE-SET <V,.>.

MOVES

1. MOVE THE LEFT TO THE RIGHT .
2. MOVE THE RIGHT TO THE LEFT .)

R2: (PRETESTS

THE SYMBOL EQUALS I .

MOVES

1. MOVE THE LEFT TO THE RIGHT .
2. MOVE THE RIGHT TO THE LEFT .
3. MOVE-FUNCTION OF THE LEFT SIGN TO THE LEFT SIGN, THE FUNCTION IS [-+,+-] .
4. MOVE-FUNCTION OF THE RIGHT SIGN TO THE RIGHT SIGN, THE FUNCTION IS [-+,+-] .)

R3: a,b. (PRETESTS

THE SYMBOL IS IN-THE-SET <V,.>.

MOVES

MOVE THE LEFT TO THE TOP-NODE .)

R4: a,b (PRETESTS

1. THE SYMBOL IS IN-THE-SET <V,.> .
2. THE SYMBOL EQUALS THE RIGHT SYMBOL .

MOVES

1. MOVE THE LEFT TO THE LEFT-LEFT .
2. MOVE THE RIGHT-LEFT TO THE LEFT-RIGHT .
3. MOVE THE RIGHT-RIGHT TO THE RIGHT .)

c,d. (PRETESTS

1. THE SYMBOL IS IN THE SET <V,.> .
2. THE SYMBOL EQUALS THE LEFT SYMBOL .

MOVES

1. MOVE THE LEFT-LEFT TO THE LEFT .
2. MOVE THE LEFT-RIGHT TO THE RIGHT-LEFT .
3. MOVE THE RIGHT TO THE RIGHT-RIGHT .)

R5: a,b. (PRETESTS

1. THE SYMBOL IS IN-THE-SET <V,.> .

MOVES

1. MOVE-FUNCTION OF THE SYMBOL TO THE SYMBOL , THE FUNCTION IS [V.,.V] .
2. MOVE-FUNCTION OF THE SIGN TO THE SIGN , THE FUNCTION IS [-+,+-] .
3. MOVE-FUNCTION OF THE LEFT SIGN TO THE LEFT SIGN , THE FUNCTION IS [-+,+-] .
4. MOVE-FUNCTION OF THE RIGHT SIGN TO THE RIGHT SIGN , THE FUNCTION IS [-+,+-] .)

R6: a,b. (PRETESTS

1. THE SYMBOL IS IN-THE-SET <V,I> .

MOVES

1. MOVE-FUNCTION OF THE SYMBOL TO THE SYMBOL , THE FUNCTION IS [VI,IV] .
2. MOVE-FUNCTION OF THE LEFT SIGN TO THE LEFT SIGN , THE FUNCTION IS [-+,+-] .)

R7: a,b. (PRETESTS

1. THE SYMBOL IS AN EXCLUSIVE-MEMBER OF <V,.> .
2. THE RIGHT SYMBOL IS AN EXCLUSIVE-MEMBER OF <V,.> .

MOVES

1. MOVE THE SYMBOL TO THE LEFT SYMBOL .
2. MOVE THE RIGHT SYMBOL TO THE SYMBOL .
3. COPY THE SYMBOL AT THE RIGHT SYMBOL .
4. MOVE THE LEFT TO LEFT-LEFT .
5. COPY THE LEFT AT THE RIGHT-LEFT .
6. MOVE THE RIGHT-LEFT TO THE LEFT-RIGHT .)

c,d. (PRETESTS

1. THE SYMBOL IS AN EXCLUSIVE-MEMBER OF <V,.> .
2. THE LEFT SYMBOL IS AN EXCLUSIVE-MEMBER OF <V,:> .
3. THE RIGHT SYMBOL EQUALS THE LEFT SYMBOL .
4. THE LEFT-LEFT EQUALS THE LEFT-RIGHT .

MOVES

1. MOVE THE LEFT SYMBOL TO THE SYMBOL .
2. MOVE THE SYMBOL TO THE RIGHT SYMBOL .
3. MOVE THE LEFT-LEFT TO THE LEFT .
4. MOVE THE LEFT-RIGHT TO THE RIGHT-LEFT .)

R8: a. (PRETESTS

THE SYMBOL EQUALS PERIOD .

MOVES

MOVE THE LEFT TO THE TOP-NODE .)

b. (PRETESTS

THE SYMBOL EQUALS PERIOD .

MOVES

MOVE THE RIGHT TO THE TOP-NODE .)

R9: (MOVES

1. MOVE THE TOP-NODE TO THE LEFT .)
2. COPY V AT THE SYMBOL .
3. COPY X AT THE RIGHT SYMBOL .)