Rand: Report on a General Problem-Solving Program

Tuesday, March 2, 2021 10:32 PM

02-gps1959

REPORT ON A GENERAL PROBLEM-SOLVING PROGRAM

A. Newell
J. C. Shaw
H. A. Simon*

P-1584

30 December 1958

Revised 9 February 1959

The RAND Corporation
1700 MAIN ST. · SANTA MONICA · CALIFORNIA

*Carnegie Institute of Technology

SUMMARY

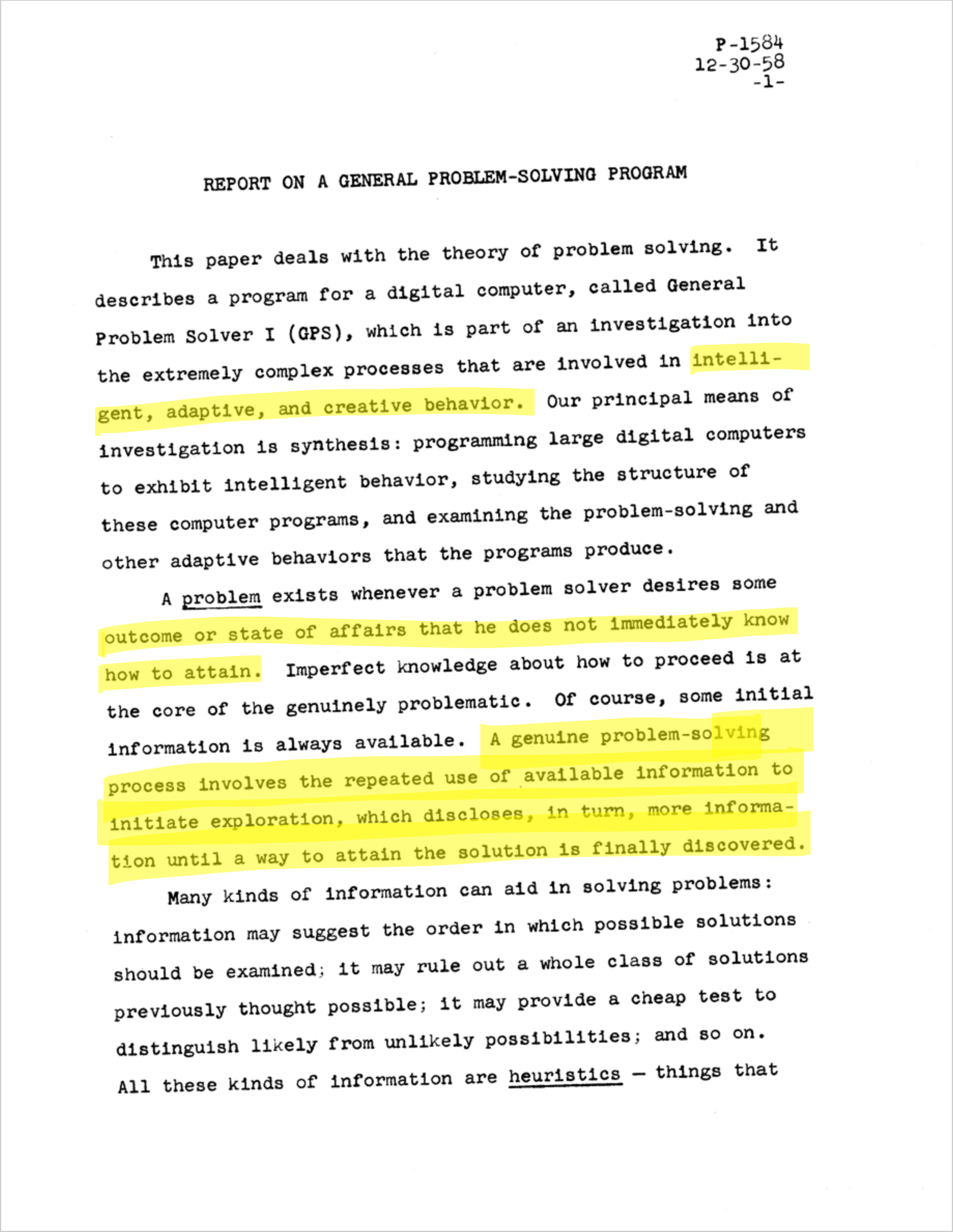
This paper reports on a computer program, called GPS-I for General Problem Solving Program I. Construction and investigation of this program is part of a research effort by the authors to understand the information processes that underlie human intellectual, adaptive, and creative abilities. The approach is synthetic — to construct computer programs that can solve problems requiring intelligence and adaptation, and to discover which varieties of these programs can be matched to data on human problem solving.

GPS-I grew out of an earlier program, the Logic Theorist, which discovers proofs to theorems in the sentential calculus. GPS-I is an attempt to fit the recorded behavior of college students trying to discover proofs. The purpose of this paper is not to relate the program to human behavior, but to describe its main characteristics and to assess its capacities as a problem-solving mechanism. The paper will present enough theoretical discussion of problem-solving activity so that the program can be seen as an attempt to advance our basic knowledge of intellectual activity. The program will be assessed from this point of view, rather than whether it offers an economical solution to a significant class of problems.

The major features of the program that are worthy of discussion are:

1. The recursive nature of its problem-solving activity.
2. The separation of problem content from problem-solving technique as a way of increasing the generality of the program.
3. The two general problem-solving techniques that now constitute its repertoire: means-ends analysis, and planning.
4. The memory and program organization used to mechanize the program (this will be noted only briefly, since there will be no space to describe the computer languages (IPL's) used to code GPS-I).

Examples will be given of how GPS solves problems in the areas of elementary symbolic logic and elementary algebra.

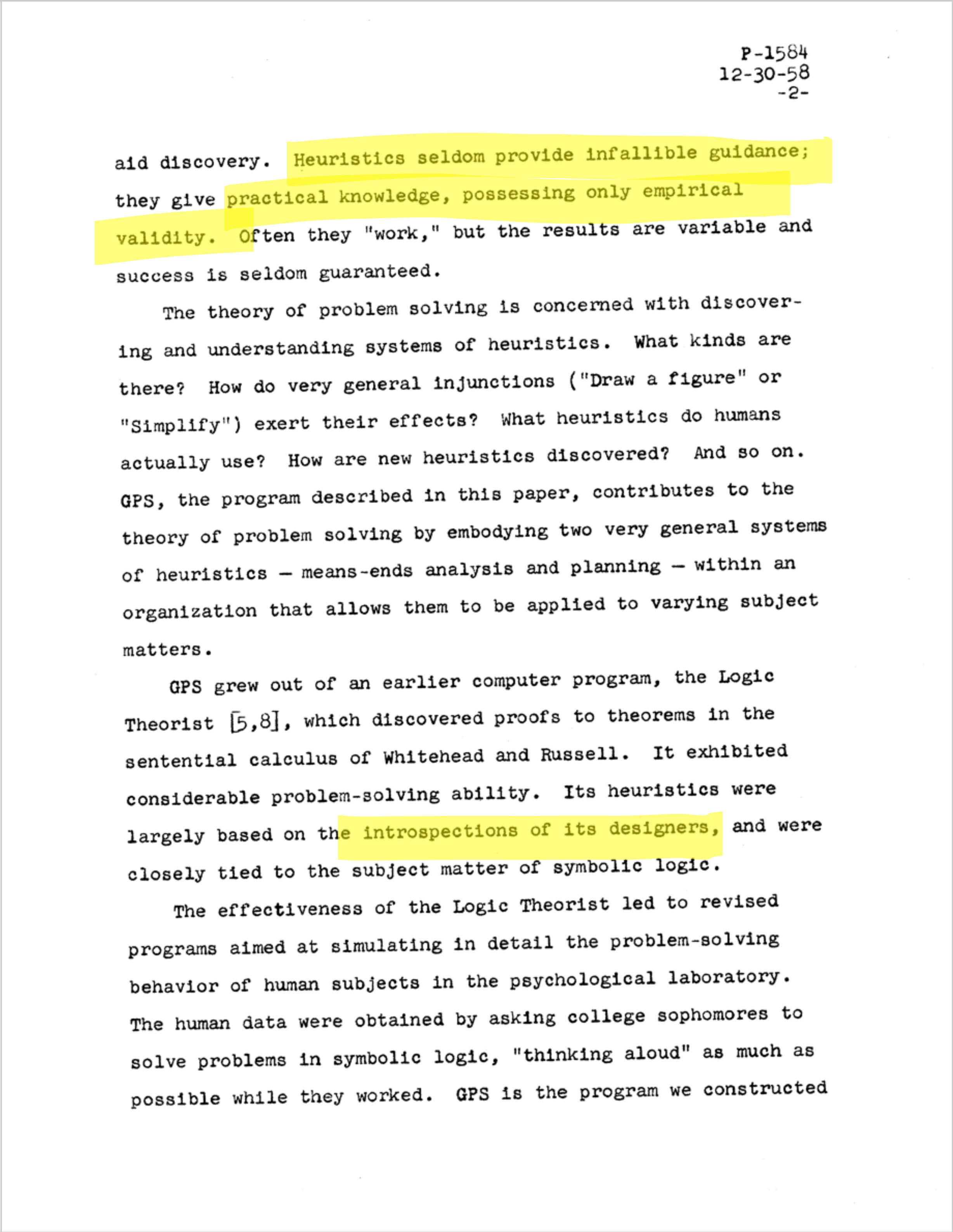


REPORT ON A GENERAL PROBLEM-SOLVING PROGRAM

This paper deals with the theory of problem solving. It describes a program for a digital computer, called General Problem Solver I (GPS), which is part of an investigation into the extremely complex processes that are involved in intelligent, adaptive, and creative behavior. Our principal means of investigation is synthesis: programming large digital computers to exhibit intelligent behavior, studying the structure of these computer programs, and examining the problem-solving and other adaptive behaviors that the programs produce.

A *problem* exists whenever a problem solver desires some outcome or state of affairs that he does not immediately know how to attain. Imperfect knowledge about how to proceed is at the core of the genuinely problematic. Of course, some initial information is always available. A genuine problem-solving process involves the repeated use of available information to initiate exploration, which discloses, in turn, more information until a way to attain the solution is finally discovered.

Many kinds of information can aid in solving problems: information may suggest the order in which possible solutions should be examined; it may rule out a whole class of solutions previously thought possible; it may provide a cheap test to distinguish likely from unlikely possibilities; and so on. All these kinds of information are *heuristics* — things that



aid discovery. Heuristics seldom provide infallible guidance; they give practical knowledge, possessing only empirical validity. Often they "work," but the results are variable and success is seldom guaranteed.

The theory of problem solving is concerned with discovering and understanding systems of heuristics. What kinds are there? How do very general injunctions ("Draw a figure" or "Simplify") exert their effects? What heuristics do humans actually use? How are new heuristics discovered? And so on. GPS, the program described in this paper, contributes to the theory of problem solving by embodying two very general systems of heuristics — means-ends analysis and planning — within an organization that allows them to be applied to varying subject matters.

GPS grew out of an earlier computer program, the Logic Theorist [5,8], which discovered proofs to theorems in the sentential calculus of Whitehead and Russell. It exhibited considerable problem-solving ability. Its heuristics were largely based on the introspections of its designers, and were closely tied to the subject matter of symbolic logic.

The effectiveness of the Logic Theorist led to revised programs aimed at simulating in detail the problem-solving behavior of human subjects in the psychological laboratory. The human data were obtained by asking college sophomores to solve problems in symbolic logic, "thinking aloud" as much as possible while they worked. GPS is the program we constructed to describe as closely as possible the behavior of the laboratory subjects as revealed in their oral comments and in the steps they wrote down in working the problems. How far it is successful in simulating the subjects' behavior — its usefulness as a psychological theory of human thinking — will be reported elsewhere [7].

We shall first describe the over-all structure of GPS, and the kinds of problems it can tackle. Then we shall describe two important systems of heuristics it employs. The first is the heuristic of means-ends analysis, which we shall illustrate with the tasks of proving theorems in symbolic logic and proving simple trigonometric identities. The second is the heuristic of constructing general plans of solutions, which we shall illustrate, again, with symbolic logic.

The Executive Program and the Task Environment

GPS operates on problems that can be formulated in terms of objects and operators. An operator is something that can be applied to certain objects to produce different objects (as a saw applied to logs produces boards). The objects can be characterized by the features they possess, and by the differences that can be observed between pairs of objects. Operators may be restricted to apply to only certain kinds of objects; and there may be operators that are applied to several objects as inputs, producing one or more objects as output (as the operation of adding two numbers produces a third number, their sum).

Various problems can be formulated in a task environment containing objects and operators: to find a way to transform a given object into another; to find an object possessing a given feature; to modify an object so that a given operator may be applied to it; and so on. In chess, for example, if we take chess positions as the objects and legal moves as the operators, then moves produce new positions (objects) from old. Not every move can be made in every position. The problem in chess is to get from a given object — the current position — to an object having a specified feature (a position in which the opponent's King is checkmated).

The problem of proving theorems in a formal mathematical system is readily put in the same form. Here the objects are theorems, while the operators are the admissible rules of inference. To prove a theorem is to transform some initial objects — the axioms — into a specified object — the desired theorem. Similarly, in the problem of integrating functions in closed form, the objects are the mathematical expressions; the operators are the operations of algebra, together with formulas that define special functions like sine and cosine. Integration in closed form is an operation that does not apply directly to every object — if it did, there would be no problem. Integration involves transforming a given object into an equivalent object that is integrable, where equivalence is defined by the set of operations that can be applied.

Constructing a computer program can also be described as a problem in these same terms. Here, the objects are possible contents of the computer memory; the operators are computer instructions that alter the memory content. A program is a sequence of operators that transforms one state of memory into another; the programming problem is to find such a sequence when certain features of the initial and terminal states are specified.

To operate generally within a task environment characterized by objects and operators, GPS needs several main components:

1. A vocabulary, for talking about the task environment, containing terms like: object, operator, difference, feature, Object #34, Operator #7.

2. A vocabulary, dealing with the organization of the problem-solving processes, containing terms like: goal type, method, evaluation, Goal Type #2, Method #1, Goal #14.

3. A set of programs defining the terms of the problem-solving vocabulary by terms in the vocabulary for describing the task environment. (We shall provide a number of examples presently.)

4. A set of programs (correlative definitions) applying the terms of the task-environment vocabulary to a particular environment: symbolic logic, trigonometry, algebra, integral calculus. (These will also be illustrated in some detail.)

Items 2 and 3 of the above list, together with the common nouns of Item 1 constitute GPS, properly speaking. Item 4 and the proper nouns of Item 1 are required to give GPS the capacity to solve problems relating to a specified subject matter. Speaking broadly, the core of GPS consists of some general, but fairly powerful, problem-solving heuristics. To apply these heuristics to a particular problem domain, GPS must be augmented by the definitions and rules of mathematics or logic that describe that domain, and then must be given a problem or series of problems to solve. The justification for calling GPS "general" lies in this factorization of problem-solving heuristics from subject matter, and its ability to use the same heuristics to deal with different subjects.

Let us look more closely at the problem-solving vocabulary and heuristics. To specify problems and subproblems, GPS has a discrete set of goal types. We shall introduce two of these initially:

Goal Type #1: Find a way to transform object a into object b. (The objects, a and b, may be any objects defined in specifying the task environment. The phrase "way to transform" implies "by applying a sequence of operators from the task environment.")

Goal Type #2: Apply operator q to object a (or to an object obtained from a by admissible transformations).

Finding a proof of a theorem (object b) from axioms (object a) is an example of a Type #1 goal; integrating (operator q) an expression (object a) is an example of a Type #2 goal.

The executive organization of GPS shown in Figure 1, is very simple. With each goal type is associated a set of methods related to achieving goals of that type. When an attempt is made to achieve a goal, it is first evaluated to see whether it is worthwhile achieving and whether achievement seems likely. If so, one of the methods is selected and executed. This either leads to success or to a repetition of the loop.

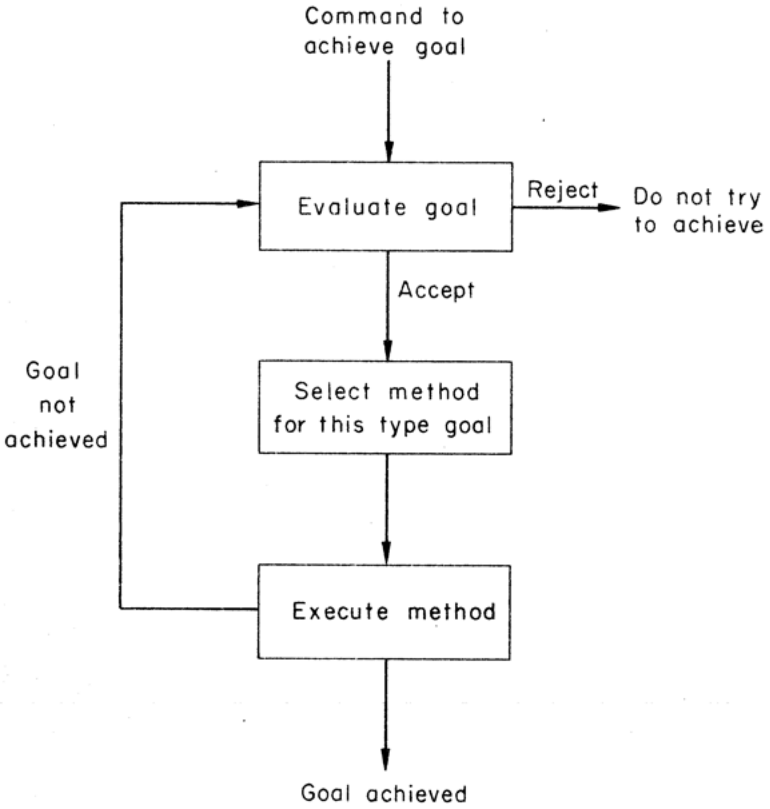
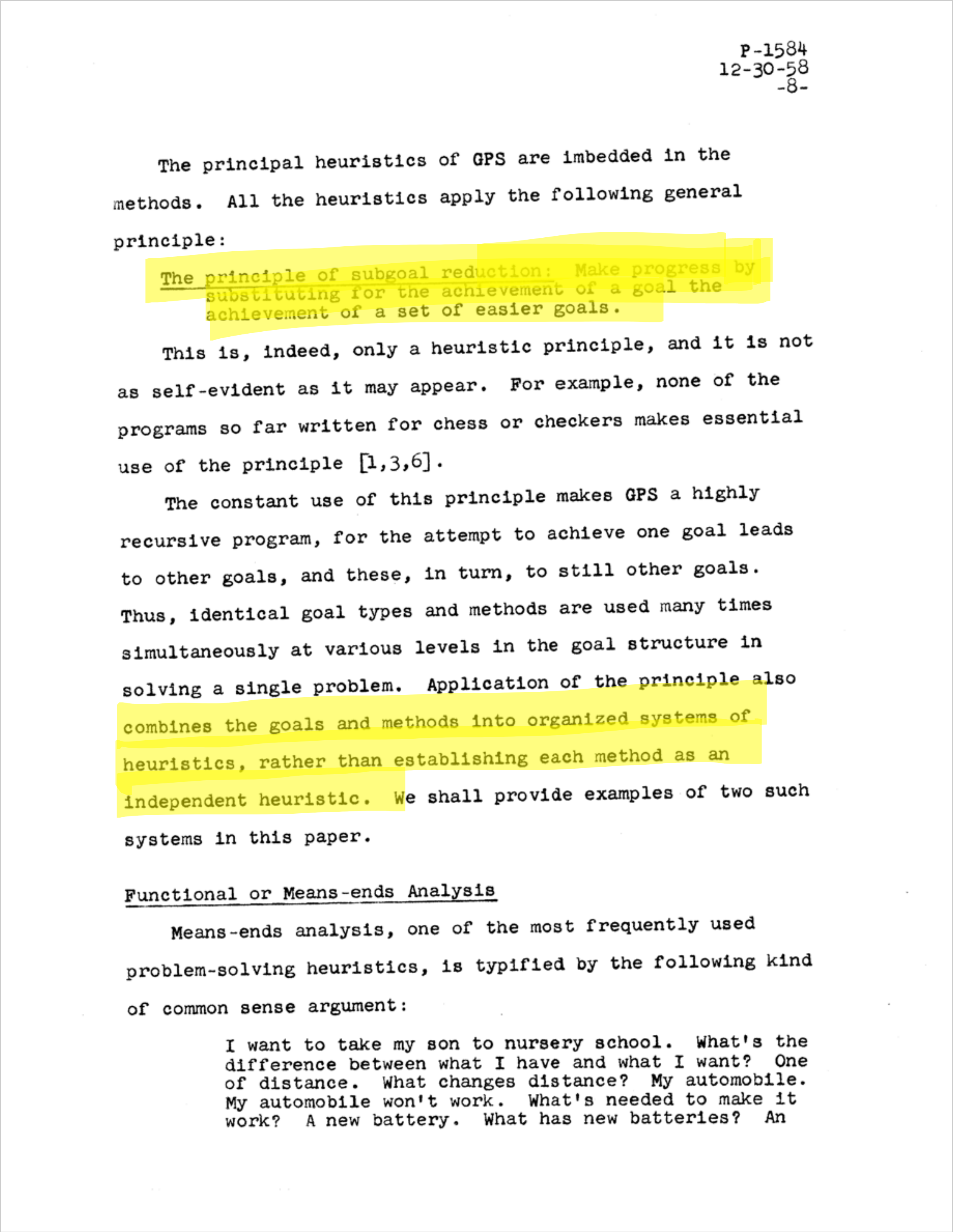


Fig. 1 — Executive organization of GPS



The principal heuristics of GPS are imbedded in the methods. All the heuristics apply the following general principle:

> The principle of subgoal reduction: Make progress by substituting for the achievement of a goal the achievement of a set of easier goals.

This is, indeed, only a heuristic principle, and it is not as self-evident as it may appear. For example, none of the programs so far written for chess or checkers makes essential use of the principle [1,3,6].

The constant use of this principle makes GPS a highly recursive program, for the attempt to achieve one goal leads to other goals, and these, in turn, to still other goals. Thus, identical goal types and methods are used many times simultaneously at various levels in the goal structure in solving a single problem. Application of the principle also combines the goals and methods into organized systems of heuristics, rather than establishing each method as an independent heuristic. We shall provide examples of two such systems in this paper.

Functional or Means-ends Analysis

Means-ends analysis, one of the most frequently used problem-solving heuristics, is typified by the following kind of common sense argument:

> I want to take my son to nursery school. What's the difference between what I have and what I want? One of distance. What changes distance? My automobile. My automobile won't work. What's needed to make it work? A new battery. What has new batteries? An

auto repair shop. I want the repair shop to put in a new battery; but the shop doesn't know I need one. What is the difficulty? One of communication. What allows communication? A telephone ... And so on.

This kind of analysis — classifying things in terms of the functions they serve, and oscillating among ends, functions required, and means that perform them — forms the basic system of heuristic of GPS. More precisely, this means-ends systems of heuristic assumes the following:

1. If an object is given that is not the desired one, differences will be detectable between the available object and the desired object.

2. Operators affect some features of their operands and leave others unchanged. Hence operators can be characterized by the changes they produce and can be used to try to eliminate differences between the objects to which they are applied and desired objects.

3. Some differences will prove more difficult to affect than others. It is profitable, therefore, to try to eliminate "difficult" differences, even at the cost of introducing new differences of lesser difficulty. This process can be repeated as long as progress is being made toward eliminating the more difficult differences.

To incorporate this heuristic in GPS, we expand the vocabulary of goal types to include:

Goal Type #3: Reduce the difference, _d_, between object _a_ and object _b_ by modifying _a_.

The core of the system of functional analysis is given by three methods, one associated with each of the three goal types, as shown in Figure 2. Method #1, associated with Goal Type #1, consists in: (a) matching the objects _a_ and _b_ to find a difference, _d_, between them; (b) setting up the Type #3

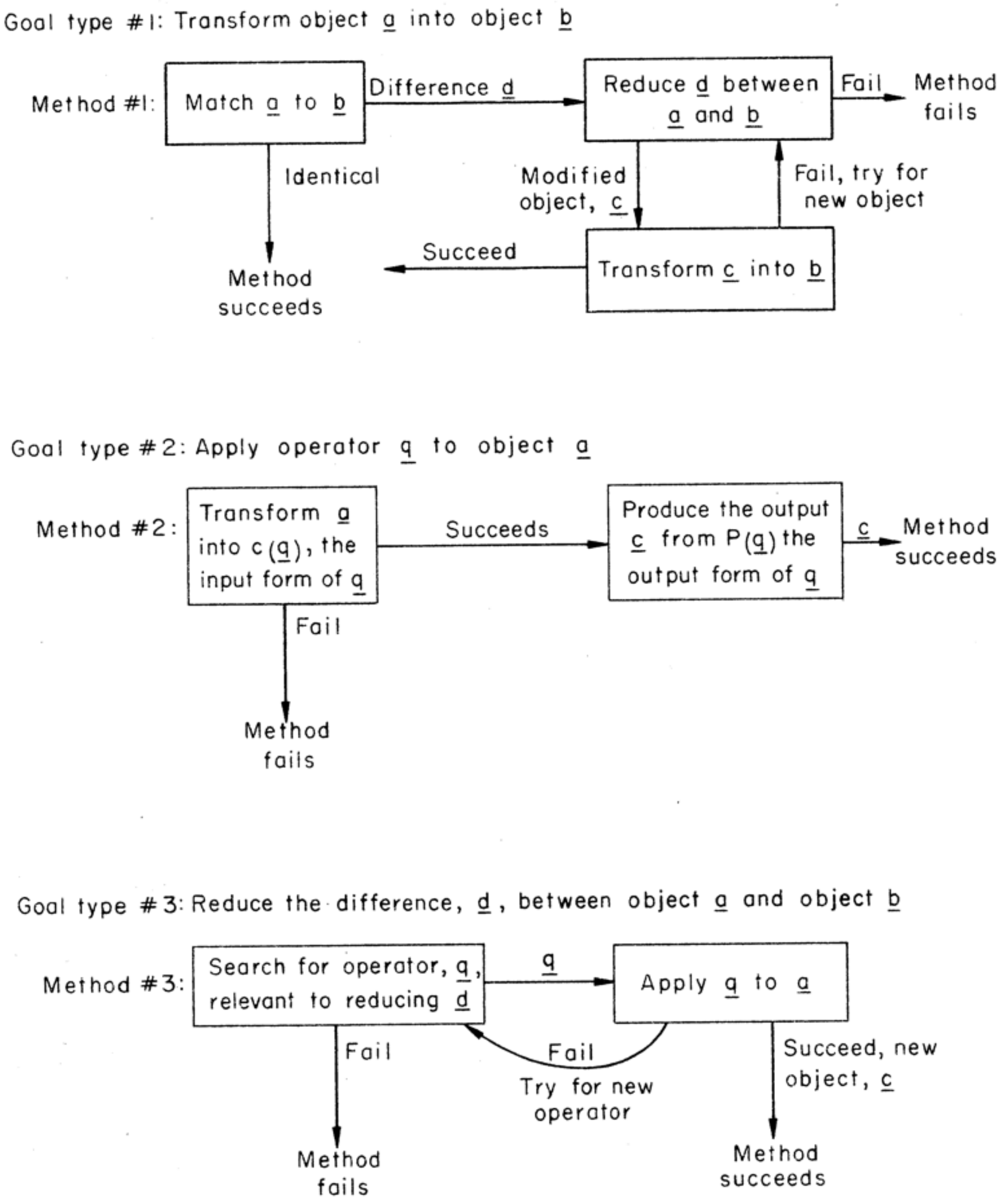


Fig. 2 — Methods for means—ends analysis

subgoal of reducing d, which if successful produces a new transformed object c; (c) setting up the Type #1 subgoal of transforming c into b. If this last goal is achieved, the original Type #1 goal is achieved. The match in step (a) tests for the more important differences first. It also automatically makes substitutions for free variables.

Method #2, for achieving a Type #2 goal, consists in: (a) determining if the operator can be applied by setting up a Type #1 goal for transforming a into the C(q), the input form of q; (b) if successful the output object is produced from P(q), the output form of q. This method is appropriate where the operator is given by two forms, one describing the input, or conditions, and the other the output, or product. The examples given in this paper have operators of this kind. Variants of this method exist for an operator given by a program, defined iteratively, or defined recursively.

Method #3, for achieving a Type #3 goal, consists in: (a) searching for an operator that is relevant to reducing the difference, d; (b) if one is found, setting up the Type #2 goal of applying the operator, which if successful produces the modified object.

Application to Symbolic Logic. This system of heuristics already gives GPS some problem solving ability. We can apply GPS to a simple problem in symbolic logic. To do so we must provide correlative definitions for objects, operators, and differences. These are summarized in Figure 3. We must also associate with each difference the operators that are relevant to modifying it. For logic this is accomplished explicitly by the table of connections in Figure 3. These connections are given to GPS, but it is not difficult to write a program that will permit GPS itself to infer the connections from the lists of operators and differences. (E.g., comparing the right side of R1 with its left side, we find they have the difference ΔP, for the symbols A and B appear in opposite orders on the two sides; hence, there is a connection between ΔP and R1.) Finally, we provide criteria of progress, in terms of a list of the differences in order of difficulty.

An illustrative logic problem and its solution are shown in Figure 4. The object, L1, is given, and GPS is required to derive the object, LO. The problem is stated to GPS in the form of a Type #1 goal: (Goal 1) Find a way to transform L1 into LO. By Figure 2, this goal type calls for Method #1. Comparison of L1 with LO shows that they have the difference, ΔP; for the "R" is on the left end of L1, but on the right end of LO. GPS now erects the Type #3 goal: (Goal 2) Reduce ΔP between L1 and LO. Goal Type #3 calls for application of Method #3. Since the table of connections (Figure 3) shows that R1 is relevant to reducing ΔP, GPS erects the Type #2 goal: (Goal 3) Apply operator R1 to L1. The reader can follow the remaining steps that lead to the solution from Figure 4.

Objects. Expressions are built up recursively from variables, P, Q, R, ..., three binary connectives, ., ⊃, v, and the unary prefix, -, called tilde. Examples of objects: P, -Q, PvQ, (-R.P) ⊃ -Q. Double tildes cancel as in ordinary algebra: --Q = Q.

Operators. There are twelve operators, given in the form $C(\underline{q}) \rightarrow P(\underline{q})$, where $C(\underline{q})$ is the input form, and $P(\underline{q})$ is the output form. Thus anything of the form at the tail of an arrow can be transformed into the corresponding expression at the head of the arrow. A double arrow means the transformation works both ways. The abstracted operators, used in the planning method, are given in the right-hand column opposite the operator.

	Operators	Abstract Operators
R1	$AvB \rightarrow BvA,\ A.B \rightarrow B.A$	Identity
R2	$A \supset B \rightarrow -B \supset -A$	Identity
R3	$AvA \leftrightarrow A,\ A.A \leftrightarrow A$	$(AA) \leftrightarrow A$
R4	$Av(BvC) \leftrightarrow (AvB)vC,\ A.(B.C) \leftrightarrow (A.B).C$	$A(BC) \leftrightarrow (AB)C$
R5	$AvB \leftrightarrow -(-A.-B)$	Identity
R6	$A \supset B \leftrightarrow -AvB$	Identity
R7	$Av(B.C) \leftrightarrow (AvB).(AvC),\ A.(BvC) \leftrightarrow (A.B)v(A.C)$	$A(BC) \leftrightarrow (AB)(AC)$
R8	$A.B \rightarrow A,\ A.B \rightarrow B$	$(AB) \rightarrow A$
R9	$A \rightarrow AvX$ (X is any expression.)	$A \rightarrow (AX)$
R10	$[A, B] \rightarrow A.B$ (Two expressions input.)	$[A, B] \rightarrow (AB)$
R11	$[A \supset B, A] \rightarrow B$ (Two expressions input.)	$[(AB), A] \rightarrow B$
R12	$[A \supset B, B \supset C] \rightarrow A \supset C$ (Two expressions input.)	$[(AB), (BC)] \rightarrow (AC)$

Figure 3a. Symbolic logic task environment

apply methods
Note differences
generate a method
reduce method to several given operators
construct several methods

Differences. The differences apply to subexpressions as well as total expressions, and several differences may exist simultaneously for the same expressions.

ΔV A variable appears in one expression that does not in the other. E.g., PvP differs by +V from PvQ, since it needs a Q; P⊃R differs by -V from R, since it needs to lose the P.

ΔN A variable occurs different numbers of times in the two expressions. E. g., P.Q differs from (P.Q)⊃Q by +N, since it needs another Q; PvP differs from P by -N, since it needs to reduce the number of P's.

ΔT There is a difference in the "sign" of the two expressions; e.g., Q versus -Q, or -(PvR) versus PvR.

ΔC There is a difference in binary connective; e.g., P⊃Q versus PvQ.

ΔG There is a difference in grouping; e.g., Pv(QvR) versus (PvQ)vR.

ΔP There is a position difference in the components of the two expressions; e. g., P⊃(QvR) versus (QvR)⊃P.

Connections between Differences and Operators. A +, -, or x in a cell means that the operator in the column of the cell affects the difference in the row of the cell. + in the first row means +V, - means -V, etc. The stars show the differences and operators that remain after abstracting, and thus marks the reduced table of connections used in the abstract task environment for planning.

			*	*			*	*	*	*	*	*
	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12
*ΔV								-	+	+	-	x
*ΔN			x				x	-	+	+	-	x
ΔT		x			x	x						
ΔC					x	x	x					
*ΔG				x			x					
ΔP	x	x										

Criteria of progress. All differences in subexpressions are less important than differences in expressions. For a pair of expressions the differences are ranked: +V, -V, +N, -N, ΔT, ΔC, ΔG, ΔP, from most important to least. E. g., ΔT is more important in -(PvQ) versus P⊃Q, but ΔC is more important in -PvQ versus P⊃Q.

Figure 3b. Symbolic logic task environment

```
Given:   L1 = R.(-P⊃Q)
Obtain:  L0 = (QvP).R

Goal 1:  Transform L1 into L0.
   Match produces position difference (ΔP).
   Goal 2:  Reduce ΔP between L1 and L0.
      First operator found is R1.
      Goal 3:  Apply R1 to L1.
         Goal 4: Transform L1 into C(R1).
            Match succeeds with A = R and B = -P⊃Q.
         Produce new object:

            L2 = (-P⊃Q).R

   Goal 5:  Transform L2 into L0.
      Match produces connective difference (ΔC) in left subexpression
      Goal 6:  Reduce ΔC  between left of L2 and left of L0.
         First operator found is R5.
         Goal 7:  Apply R5 to left of L2.
            Goal 8:  Transform left of L2 into C(R5).
               Match produces connective difference (ΔC) in left subexpression.
               Goal 9:  Reduce ΔC between left of L2 and C(R5).
                  Goal rejected: difference is no easier than difference in
                  Goal 6.
         Second operator found is R6.
         Goal 10:  Apply R6 to left of L2.
            Goal 11:  Transform left of L2 into C(R6).
               Match succeeds with A = -P and B = Q.
            Produce new object:

               L3 = (PvQ).R

   Goal 12:  Transform L3 into L0.
      Match produces position difference (ΔP) in left subexpression.
      Goal 13:  Reduce ΔP between left of L3 and left of L0.
         First operator found is R1.
         Goal 14:  Apply R1 to left of L3.
            Goal 15:  Transform left of L3 into C(R1).
               Match succeeds with A = P and B = Q.
            Produce new object:

               L4 = (QvP).R

      Goal 16:  Transform L4 into L0.
         Match shows L4 is identical with L0, QED.
```

Figure 4. Example of means-ends analysis in logic.

The resulting derivation may be summarized:

Object	Operator
L1 = R. (-P⊃Q)	
L2 = (-P⊃Q).R	Apply R1 to L1
L3 = (P v Q).R	Apply R6 to left side of L2
L4 = (Q v P).R	Apply R1 to left side of L3
L4 is identical with LO	Q. E. D.

GPS can solve problems a good deal more difficult than the simple one illustrated. To make full use of the twelve operators, an additional method is added to the Type #3 goal that searches the available objects for the additional input required in rules R10, R11, and R12.

Application to Trigonometry. GPS is a general problem solver to the extent that its heuristics can be applied to varying subject matters, given the appropriate correlative definitions. Elementary algebra and calculus provide a subject matter distinct from logic, and Figure 5 shows the fragment of this task environment necessary for GPS to try to prove some simple trigonometric identities. The objects are now algebraic and trigonometric expressions; and the operators perform factorization, algebraic simplification, and trigonometric transformation. The differences are the same as in logic, except for two omissions, which are related to the associative and commutative laws. In logic these must be performed explicitly, whereas in ordinary algebra a notation is used that makes these laws implicit and their operation automatic. The connections between differences and operators is not made via a simple table, as in logic, but requires a

Objects. Ordinary algebraic expressions, including the trigonometric functions. The associative and commutative laws are implicit in the notation: the program can select freely which terms to use in an expression like $(x + y + z)$.

Operators.

A0 Combine: Recursively defined to apply the following elementary identities from the innermost subexpressions to the main expression:

$A + (B+C) \longrightarrow A + B + C$, $A(BC) \longrightarrow ABC$,

$A + 0 \longrightarrow A$, $A + A \longrightarrow 2A$, $A - A \longrightarrow 0$

$A0 \longrightarrow 0$, $A1 \longrightarrow A$, $AA \longrightarrow A^2$, $A^BA^C \longrightarrow A^{B+C}$

$A^0 \longrightarrow 1$, $0^A \longrightarrow 0$, $A^1 \longrightarrow A$, $(A^B)^C \longrightarrow A^{BC}$

A1 $(A-B)(A+B) \longleftrightarrow A^2-B^2$

A2 $(A+B)^2 \longleftrightarrow A^2 + 2AB + B^2$

A3 $A(B+C) \longleftrightarrow AB + AC$

T1 $\tan x \longleftrightarrow 1/\cot x$

T2 $(\tan x)(\cot x) \longleftrightarrow 1$

T3 $\tan x \longleftrightarrow \sin x / \cos x$

T4 $\cot x \longleftrightarrow \cos x / \sin x$

T5 $\sin^2 x + \cos^2 x \longleftrightarrow 1$

Differences. Defined as in logic: ΔV, ΔN, ΔC, ΔT. ΔG and ΔP do not occur in algebra, since associativity and commutativity are built into the programs for handling expressions. The trigonometric functions are detected by ΔV and ΔN.

Connections between Differences and Operators, A + , -, or x in a cell means that the operator in the column of the cell affects the difference in the row of the cell. A t means that the test defined at the bottom is applied.

	A0	A1	A2	A3	T1	T2	T3	T4	T5
ΔV	-				t	t	t	t	t
ΔC	x	x	x	x					
ΔN	-	x	x	x	t	t	t	t	t
ΔT	x								

Test t: accept if other functions in output form already occur in expression.

Criteria of progress. Defined as in logic, but with ΔC more important than ΔN or ΔP.

Figure 5. Trigonometry task environment.

comparison of the object with the output form of the operator. The criteria of progress remain the same as for logic.

GPS can now attempt to prove a trigonometric identity like:

$$(\tan x + \cot x)\sin x \cos x = 1$$

This is given as the problem of tranforming the left side, which becomes L1, into the right side, L0. The process of solving the problem, which involves 33 goals and subgoals, is shown in Figure 6, which is to be interpreted in exactly the same way as Figure 4, with the help of Figures 2 and 5, except that explicit mention of the methods is suppressed.

Planning as a Problem-solving Technique

The second system of heuristic used by GPS is a form of planning that allows GPS to construct a proposed solution in general terms before working out the details. It acts as an antidote to the limitations of means-ends analysis in seeing only one step ahead. It also provides an example of the use of an auxiliary problem in a different task environment to aid in the solution of a problem.* Planning is incorporated in GPS by adding a new method, Method #4, to the repertoire of the Type #1 goal.

This Planning Method (see Figure 7) consists in (a) abstracting by omitting certain details of the original objects

*See the work of H. Gelernter and N. Rochester on theorem proving programs for plane geometry [2], where the geometric diagram provides an example of a very powerful auxiliary problem space.

```
Given:   L1 = (tan x + cot x) sin x cos x
Obtain:  L0 = 1

Goal 1: Transform L1 into L0.
   Goal 2: Reduce -V between L1 and L0 (tan).
      Goal 3: Apply A0 (combine) to L0 [no change produced] .
      Goal 4: Apply T1 to L1.
         Goal 5: Transform L1 into C(T1)  succeeds .

         L2 = [(1/cot x) + cot x]sin x cos x

   Goal 6: Transform L2 into L0.
      Goal 7: Reduce -V between L2 and L0 (cot).
         Goal 8: Apply A0 to L2 [no change produced] .
         Goal 9: Apply T4 to L2.
            Goal 10: Transform L2 into C(T4) [succeeds] .

            L3 = [(1/(cos x/sin x)) + (cos x/sin x)] sin x cos x

      Goal 11: Transform L3 into L0.
         Goal 12: Reduce -V between L3 and L0 (cos).
            Goal 13: Apply A0 to L3:

            L4 = [(sin x/cos x) + (cos x/sin x)] sin x cos x

      Goal 14: Transform L4 into L0.
         Goal 15: Reduce -V between L4 and L0 (sin).
            Goal 16: Apply A0 to L4 [no change produced].
            Goal 17: Apply T5 to L4.
               Goal 18: Transform L4 into C(T5).
                  Goal 19: Reduce ΔC between L4 and C(T5) (. to +).
                     Goal 20: Apply A0 to L4 [no change produced].
                     Goal 21: Apply A1 to L4.
                        Goal 22: Transform L4 into C(A1).
                           Goal 23: Reduce ΔC between L4 and C(A1) [reject] .
                     Goal 24: Apply A3 to L4.
                        Goal 25: Transform L4 into C(A3) [succeeds].

                        L5 = [sin x/cos x]sin x cos x + [cos x/sin x] sin x cos x

                  Goal 26: Transform L5 into C(T5).
                     Goal 27: Reduce ΔC between left of L5 and left of C(T5).
                        Goal 28: Apply A0 to left of L5:

                           L6 = sin$^2$x + [cos x/sin x] sin x cos x

                     Goal 29: Transform L6 into C(T5).
                        Goal 30: Reduce ΔC between right of L6 and right of C(T5).
                           Goal 31: Apply A0 to right of L6:

                              L7 = sin$^2$x + cos$^2$x

                        Goal 32: Transform L7 into C(T5) [succeeds].

                        L8 = 1

         Goal 33: Transform L8 into L0 [identical], QED.
```

Figure 6. Example of means-ends analysis on trigonometry.

and operators, (b) forming the corresponding problem in the abstract task environment, (c) when the abstract problem has been solved, using its solution to provide a plan for solving the original problem, (d) translating the plan back into the original task environment and executing it. The power of the method rests on two facts. First, the entire machinery of GPS can be used to solve the abstract problem in its appropriate task environment; and, because of the suppression of detail, this is usually a simpler problem (having fewer steps) than the original one. Second, the subproblems that make up the plan are collectively simpler (each having few steps) than the original problem. Since the exploration required to solve a problem generally increases exponentially with the number of steps in the solution, replacement of a single large problem with several smaller problems, the sum of whose lengths is about equal to the length of the original problem, may reduce the problem difficulty by whole orders of magnitude.

Figure 8 shows the Planning Method applied to a problem of symbolic logic. The particular abstraction scheme that is illustrated ignores differences among connectives and the order of symbols (ΔC and ΔP), replacing, for example, "(R⊃-P) · (-R⊃Q)" with "(PR) (QR)." The operators are similarly abstracted, so that "AvB→BvA" becomes "(AB)→(AB)" — i.e., the identity operator — and "A.B→A" becomes "(AB)→A," as shown in Figure 3. The abstracted problem, Transform A1 into AO, has several solutions in the abstracted task environment. One of

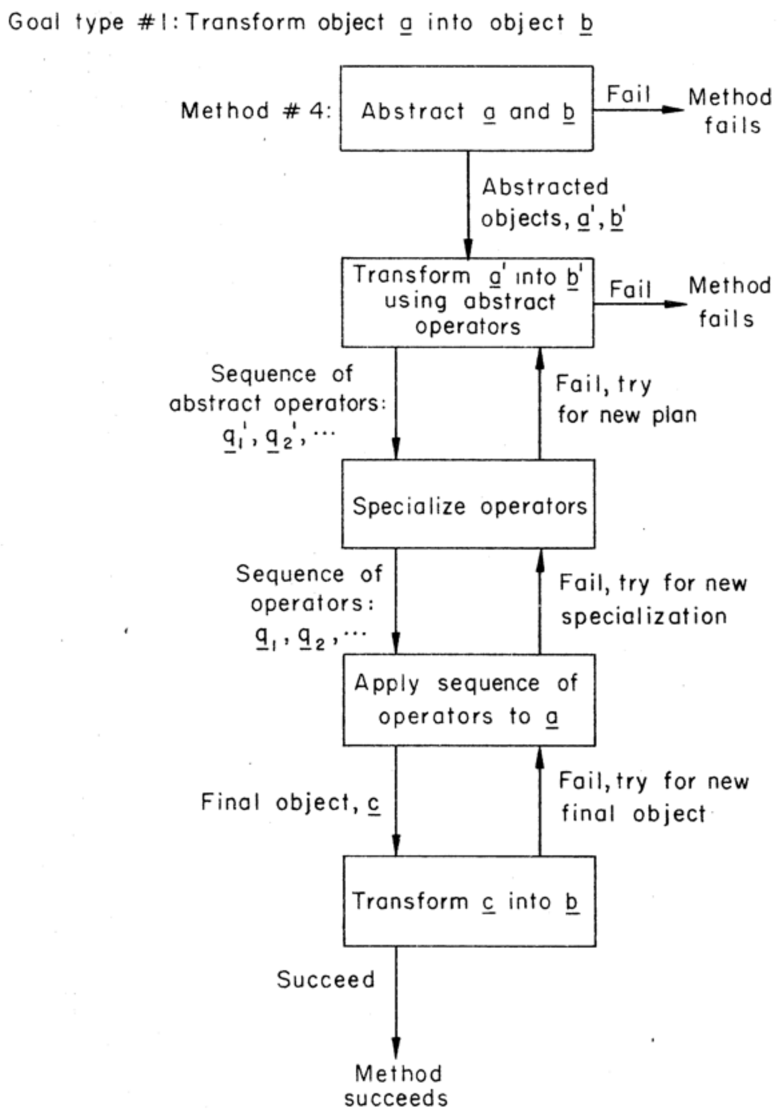
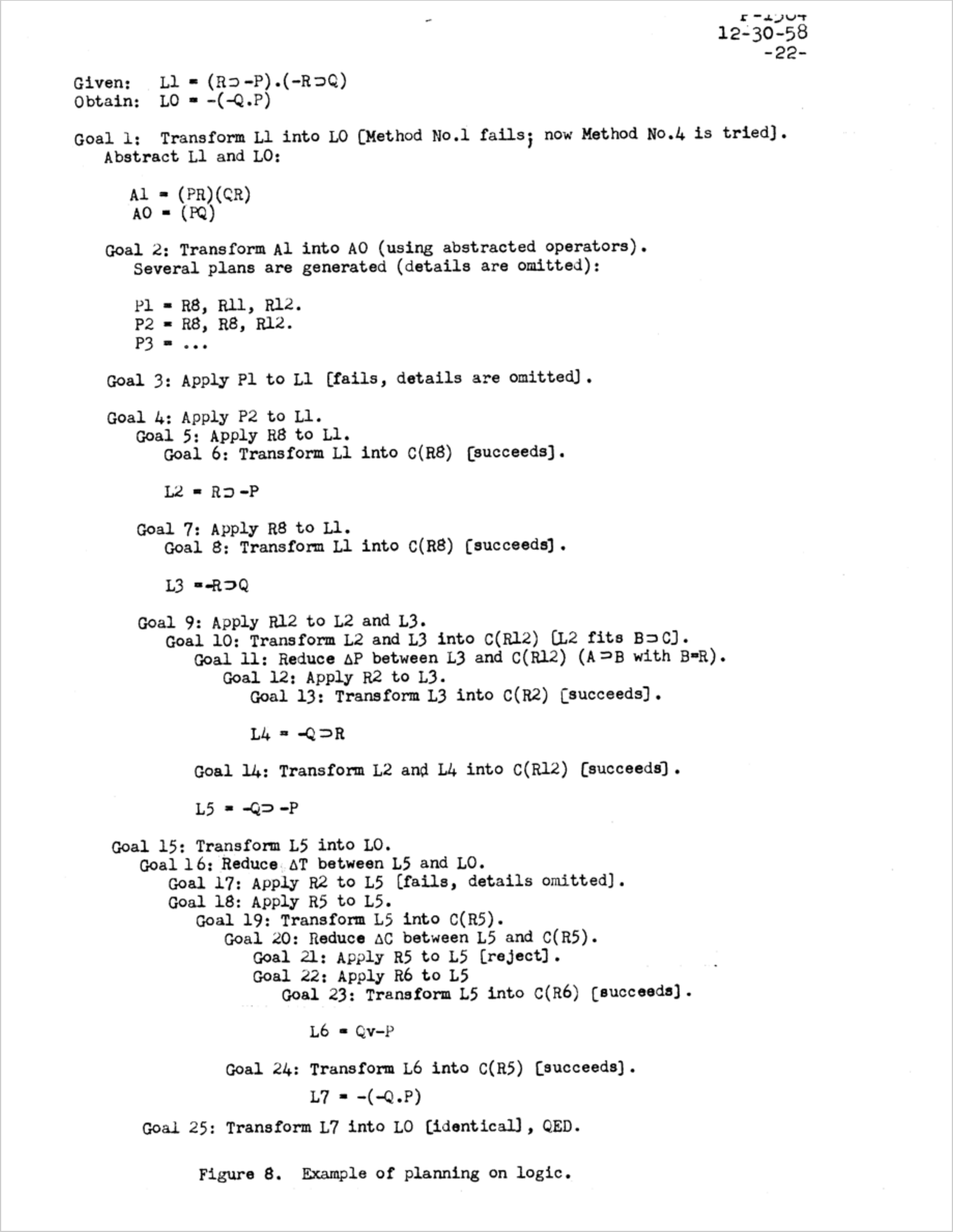

Goal type #1: Transform object a into object b
Method # 4:
Abstract a and b
Fail
Method fails
Abstracted objects, a', b'
Transform a' into b' using abstract operators
Fail
Method fails
Sequence of abstract operators: q_1', q_2', ···
Fail, try for new plan
Specialize operators
Sequence of operators: q_1, q_2, ···
Fail, try for new specialization
Apply sequence of operators to a
Final object, c
Fail, try for new final object
Transform c into b
Succeed
Method succeeds

Fig. 7—Planning method



```
Given:   L1 = (R⊃-P).(-R⊃Q)
Obtain:  L0 = -(-Q.P)

Goal 1:  Transform L1 into L0 [Method No.1 fails; now Method No.4 is tried].
   Abstract L1 and L0:

      A1 = (PR)(QR)
      A0 = (PQ)

   Goal 2: Transform A1 into A0 (using abstracted operators).
      Several plans are generated (details are omitted):

      P1 = R8, R11, R12.
      P2 = R8, R8, R12.
      P3 = ...

   Goal 3: Apply P1 to L1 [fails, details are omitted].

   Goal 4: Apply P2 to L1.
      Goal 5: Apply R8 to L1.
         Goal 6: Transform L1 into C(R8) [succeeds].

         L2 = R⊃-P

      Goal 7: Apply R8 to L1.
         Goal 8: Transform L1 into C(R8) [succeeds].

         L3 = -R⊃Q

      Goal 9: Apply R12 to L2 and L3.
         Goal 10: Transform L2 and L3 into C(R12) [L2 fits B⊃C].
            Goal 11: Reduce ΔP between L3 and C(R12) (A⊃B with B=R).
               Goal 12: Apply R2 to L3.
                  Goal 13: Transform L3 into C(R2) [succeeds].

                  L4 = -Q⊃R

            Goal 14: Transform L2 and L4 into C(R12) [succeeds].

            L5 = -Q⊃-P

   Goal 15: Transform L5 into L0.
      Goal 16: Reduce ΔT between L5 and L0.
         Goal 17: Apply R2 to L5 [fails, details omitted].
         Goal 18: Apply R5 to L5.
            Goal 19: Transform L5 into C(R5).
               Goal 20: Reduce ΔC between L5 and C(R5).
                  Goal 21: Apply R5 to L5 [reject].
                  Goal 22: Apply R6 to L5
                     Goal 23: Transform L5 into C(R6) [succeeds].

                        L6 = Qv-P

               Goal 24: Transform L6 into C(R5) [succeeds].
                        L7 = -(-Q.P)

      Goal 25: Transform L7 into L0 [identical], QED.
```

Figure 8. Example of planning on logic.

these may be summarized:

Object	Operation
A1 = (PR) (QR)	
A2 = (PR)	Apply R8 to get left side of A1
A3 = (QR)	Apply R8 to get right side of A1
(PQ)	Apply R12 to A2 and A3
But (PQ) is identical with A0	Q. E. D.

Transforming A1 into A0 is the abstract equivalent of the problem of transforming L1 into L0. The former is solved by applying the abstracted operators corresponding to R8, R8, and R12 in sequence. Hence (Figure 8, Goal 4) a plan for solving the original problem is to try to apply R8 to L1 (obtaining a new object whose abstract equivalent is (PR)), applying R8 to the other side of L1 (obtaining an object corresponding to (QR)), applying R12 to the objects thus obtained, and finally, transforming this new object (which should be an abstract equivalent of L0) into L0. Each of the first three parts of this plan constitutes a Type #2 goal in the original task environment — that is, it requires the application of a specified operator to a specified expression. These three Type #2 goals are Goals 5, 7, and 9, respectively, in Figure 8. The first two are achieved almost trivially, and the third requires only five subgoals. The result is the expression, L5, which is to be transformed to L0, as indicated by a Type #1 goal (Goal 15). Note that achieving the plan only involves operators that become the identity operator upon abstraction.

Like the other heuristics, the planning heuristic offers no guarantees that it will always work. It may generate no plan, a single plan, or several. More serious, a plan may turn out to be illusory — it may prove impossible to carry it out. The first plan generated in the illustrative problem, for example — R8, R11, R12 — does not provide a basis for a valid derivation when it is translated back to the concrete task environment. The time wasted in fruitless efforts to execute invalid plans must be counted in evaluating the planning heuristic, but even when allowance is made for this cost, the heuristic remains a very powerful one.

Conclusion

We have now described the principal heuristics that are used by GPS, and have shown how these heuristics enable it to solve problems. Although limitations of space prevent full discussion, we would like to conclude by mentioning several aspects of GPS that have received inadequate treatment in this paper.

1. GPS requires additional goals and methods, and we have investigated a number of these to varying degrees. For example, in many problems features of objects come to play a larger role than differences between objects — e.g., control of center in chess. A goal and methods similar to Type #3 are needed to handle features. Solving single equations in one variable also requires a new goal and new methods embodying such heuristics as operating on

both sides of an equation to isolate the variable. This goal is already needed in trigonometry to recognize that $\sin^4 x - \cos^4 x$ is of the form $y^2 - z^2$, so it can be factored. Already with its present repertoire GPS can handle algebraic simplification, and integration of relatively simple expressions with the aid of a table of integrals. There is some hope, as the example introducing means-ends analysis indicates, that GPS may be capable of the common sense thinking that gets most humans through the details of each day's living.

2. GPS is given very little information about a task environment, and deals with the most general features of it. Beyond a point most of the heuristics of GPS will be devoted to discovering special systems of heuristics for particular subject matters. In trigonometry, for example, GPS needs to be able to learn such heuristics as "reduce everything to sines and cosines," and "follow a trigonometric step with an algebraic step." In fact it followed these roughly in the example given, but only in a cumbersome fashion. The one narrow piece of learning we have mentioned — associating differences with operators — is a start in this direction.

weak method

3. Realizing programs like GPS on a computer is a major programming task. Much of our research effort has gone into the design of programming languages (information processing languages) that make the writing of such programs practicable. We must refer the reader to other publications for a description

of this work [4,9]. However, we would like to emphasize that our description of GPS in this paper ignores all information handling problems: how to keep track of the goals; how to associate with them the necessary information, and retrieve it; how to add methods to an already running system; and so on. These technical problems form a large part of the problem of creating intelligent programs.

4. In this paper we have also underemphasized the role of the evaluation step — the opportunity to reject a goal before any effort is spent upon it. The methods are generative, producing possible solutions. They are only heuristic, and so will produce many more possibilities than can be explored. The evaluation applies additional heuristics to select the more profitable paths, and strongly affects the problem solving ability of GPS. The means-ends analysis is a general heuristic because progress can be evaluated in the same general terms as the methods. More specific evaluations will again require learning.

5. Limitations of space have forced our examples to focus on the correct solution path. They do not convey properly the amount of selection, and trial and error. This is particularly unfortunate, since dramatically viewed, problem-solving is the battle of selection techniques against a space of possibilities that keeps expanding exponentially [5,7].

REFERENCES

1. Bernstein, A. et al., A Chess-playing Program for the IBM 704, *Proceedings of the 1958 Western Joint Computer Conference*, May 1958.

2. Gelernter, H. L. and N. Rochester, Intelligent Behavior in Problem-solving Machines, *IBM Journal of Research and Development*, Vol. 2, No. 4, October 1958.

3. Kister, J. et al., Experiments in Chess, *Journal of the Association for Computing Machinery*, Vol. 4, No. 2, April 1957.

4. Newell, A. and J. C. Shaw, Programming the Logic Theory Machine, *Proceedings of the 1957 Western Joint Computer Conference*, February 1957.

5. Newell, A., J. C. Shaw, and H. A. Simon, Empirical Explorations of the Logic Theory Machine, *Proceedings of the 1957 Western Joint Computer Conference*, February 1957.

6. Newell, A., J. C. Shaw, and H. A. Simon, Chess-playing Programs and the Problem of Complexity, *IBM Journal of Research and Development*, Vol. 2, No. 4, October 1958.

7. Newell, A., J. C. Shaw, and H. A. Simon, *The Processes of Creative Thinking*, The RAND Corporation, Paper P-1320, August 1958.

8. Newell, A. and H. A. Simon, The Logic Theory Machine, *Transactions on Information Theory*, IT-2, No. 3, September 1956.

9. Shaw, J. C. et al., A Command Structure for Complex Information Processing, *Proceedings of the 1958 Western Joint Computer Conference*, May 1958.