Rough Draft

CIP Working Paper #7
December 29, 1957

Preliminary Description of
General Problem Solving Program - I (GPS-I)

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

GPS is a computer program for reasoning about a broad class of problems. The class can be characterized, roughly, as problems involving manipulation of of a set of "objects" by a set of operations in order to achieve a desired object or configuration of objects. This seems a sufficiently broad class of problems to justify the name GPS-I. However, the class of problems that the program can successfully reason about - either solve, or make a good try at solving - is not really known in any useful way. There are a set of some 20 processes (the V's) that constitute the entire knowledge GPS must have about the subject matter. In essence, the V's form a set of parameters of the program, and any subject matter that can be characterized by the V's can be reasoned about by GPS. Conversely, until an attempt is made to characterize a subject matter by the V's it is unknown whether GPS can reason about it or not.

Historically, this program was not developed as a general problem solving program, but for the specific purpose of describing humans solving problems in symbolic logic. Its present form evolved from trying to keep the problem solving process distinct from the concern with the particluar features of symbolic logic.

The purpose of this working paper is to provide an overview of the system and a guide to the program. We will not discuss any of the features or issues involved in what kinds of problems GPS might solve, or how it could be different in various ways.

II. Overview of the system

Task environment. The environment GPS faces consists of objects, and operators that modify these objects, or produce new objects from old. The tasks one can set GPS are to attain a set of objects of specified nature from some given set of initial objects plus certain operations. GPS does not deal directly with the objects of the environment, but with expressions - i.e., symbolic representations of the objects. In symbolic tasks, like logic or algebra, the objects may be the expressions themselves. In any event, all of the structure of the program will be described in terms of expressions.

Goals and goal structure. The main unit of organization in GPS is the goal. A goal is a symbol that associates together information necessary to direct the behavior of the program in a given way - i.e., towards "achievement" of the goal. There are various general types of goals (five, at the moment) corresponding to uniformities in the end to be achieved, and in the way of going about it. Currently we have

Type 0: find a way of getting from a to b;

Type 1: reduce the difference d between a and b;

Type 2: apply the operator q to a;

Type 3; achieve at least one of a set of goals ("or" goal);

Type 4: achieve all of a set of goals ("and" goal).

The a and b refer ultimately to objects, the d to differences that GPS can perceive between objects, and the q to the operators that modify objects.

Associated with each goal are four kinds of information:

the goal type;

a set of goal components, which correspond to the a, b, d, and q in the above statements of goal types;

a set of links to other goals, objects, etc., that allow association over the network of goals;

miscellaneous data, such as the order of generation of a goal, previous history of solution attempts, etc..

A goal component is a symbol that associates together the information necessary to specify an entity (to use a general term) that enters into the specification of a goal. So far the entities required are expressions, which ultimately refer objects; differences between expressions; and operators that modify expressions. Goal components that specify homogeneous sets of these entities are also admitted. In the case of differences or operators no information is needed other than the name of the difference or operator. In the case of expressions however, a goal component has several parts:

the name of the total expression being referenced;

a program for finding the subexpression referenced;

a list of variables in the total expression that have been specified and the expressions that are to be substituted for them.

These will be covered in more detail when we cover the nature of expressions and the assumptions made about them.

Methods. Each goal has associated with it, via the goal type, a set of methods for achieving the goal. Thus all goals of the same type have the same repertoire of methods. It is conceivable that a goal type could be so specific that there would be a single method which simply accomplished the goal without further ado. In general, however, things are not so easy, and the methods provide ways of constructing subgoals whose accomplishment leads to accomplishment of the original goal (the supergoal). Thus the methods for a set of goal types form a system of recursive prescriptions of the behavior of GPS for each goal type in terms of other goals of other types. The recursive system is terminated either by success - the accomplishment of some goals rather than further reduction to subgoals - or by evaluation processes that signal failure of a particular

reduction to subgoals to yield a successful line of attack (more about this below).

A rough sketch of the three methods programmed so far for goal types 0, 1, and 2 will show this recursive interdependence very well:

> Method for type 0 goal (find a way of getting from a to b).
>
> Match a to b, component by component, and if not identical note the difference, d. Try to accomplish the type 1 subgoal: reduce the difference d between a and b. If this goal is achieved, it yields a new expression a'. Now try to accomplish the type 0 subgoal: find a way of getting from a' to b.
>
> Method for type 1 goal (reduce the difference d between a and b).
>
> Search for an operator q that is relevant to reduction of the difference d. Set up the type 2 subgoal: apply q to a. If this works, q (a) is the modified expression a'; if it doesn't work, search for another operator and try again.
>
> Method for type 2 goal (apply operator q to a).
>
> See if the conditions are met for applying the operator. If they are, apply it, and all OK. If not, note the difference d between what a is and what it should be to apply q. Try to accomplish the type 1 subgoal: reduce the difference d between a and c, where c is an expression for what a should be like. If the goal is achieved, it yields a new expression a'. (Note: this method is applicable only for operators that provide an expression c.)

Thus, a type 0 goal leads to subgoals of types 0 and 1; a type 1 goal leads to subgoals of type 2; and a type 2 goal leads to subgoals of types 1 and 2. There may be more than one method for a given goal type, so that the recursive system above is illustrative, and not exhaustive.

Evaluations. To each goal type corresponds a process which evaluates whether the goal of that type should be worked on, and if so, selects which method will be used. This process (R10 through R14 for types 0 through 4 respectively) epitomizes the goal type, and when we said earlier that each goal had associated to it its goal type, the association is actually to the evaluation process R10 - R14.

Behavior cycle. The major cycle of behavior of GPS is exceedingly simple, and correspondingly uninformative. There is a special cell, WO, which holds the current goal - i.e., the one that is currently guiding behavior. Goals are formed only by methods, and always have either a positive or negative outcome (accompanied by various outputs, perhaps) if they are attempted. Thus, in the middle of a method, a goal may be formed and put in WO. Then the evaluation process for its goal type is executed. Either there is an immediate return of "fail," in which case control remains in the current method; or some method is selected for attempting the newly set-up goal, in which case control passes to the new method at the lower level. Eventually, of course, control returns to the current method accompanied by the signal of success or failure for the goal that the method set up and attempted. And eventually also, the current method will terminate with a similar signal of success or failure to the next higher level.

Although there will always exist a highest executive which set up the first goal, there is no built-in guarantee that the control will ever return to it in finite time, nor that it will terminate the total program even if it does get control back. Whether control ever comes back depends on the success of the methods, and on the severity of the evaluation routines in rejecting unpromising leads. And the behavior of this highest executive - which is the one program, not a method, which can set up goals - is not specified at all by the scheme laid out here.

Expressions. Earlier it was mentioned that objects are represented for GPS by expressions, so that operations performed on objects induce corresponding operations perfotmed on expressions. The exact nature of objects, and hence expressions, is one of the main points of contact of GPS with the subject matter it is reasoning about. Hence, only a minimal set of properties of expression are specified, in order to give the greatest possible scope to the program. Most of the V's are

processes for manipulating expressions or gaining information from them, and hence, the V's imply a set of properties that expressions must have. Also, the structure of the expression component implies properties that expressions must possess. The following list gives a reasonably complete definition of an expression:

> Expressions are list structures, and are capable of being filed away, copied, and erased. (A large proportion of the V's are concerned with just such housekeeping operations.)
>
> Expressions have structure - that is, they are built up from subexpressions in a hierarchy of parts and subparts.
>
> Expressions have variables in them, for which other expressions can be substituted. This is consonant with the part-subpart character referred to above.
>
> It is possible to distinguish the "structure" of an expression from the particular symbols that fill it out and give it "content." Thus there is a way of putting two expressions in correspondence, even if they are not identical. (VO is a process for providing this correspondence.)

Differences. We have talked frequently above about differences without defining them. Given that two expressions are brought into correspondence, they can fail to be identical in various ways. The set of possible differences, and their relevance to the problem at hand is clearly dependent on subject matter. GPS only assumes that some set of differences are given - i.e., can be detected in a comparison (or match) of two expressions. It does assume that the same set of differences is applicable to all subexpressions in a given expression - or at least that it can delegate the issue of non-applicability to the differences themselves.

As an entity in GPS, a difference is a symbol which associates together the various kinds of information listed below. It is assumed part of the subject matter and is therefore given to GPS via one of the V's, V2. The components of a difference are:

> a test whether the difference holds between two expressions;
>
> a process that generates operators related to the difference - i.e.,

operators with some chance of affecting the difference;

links to goals that have used this difference;

miscellaneous data about experience with the difference.

Operators. The various operators that modify expressions or produce new ones are also part of the subject matter. As with the difference, the operator is just a symbol that associates together information and processes relevant to the operator. At the moment, all that is needed is a process for trying the operator; if it works, producing the product, and if it doesn't producing a difference. An extremely common way to specify operations on expressions is to provide forms - one form to identify the relevant parts of the expressions and see if the operator is even applicable, and another form to give the product. For example, the commutative law of algebra is usually given as the statement that any expression of form A + B can be transformed into B + A. In this case the operator is really replaced by an expression (or pair of them). GPS is prepared to have its operators given in this way also.

III. Summary of conventions

We give here a few conventions used in the coding and describing of GPS. GPS is coded in IPL-IV, and familiarity with it will be assumed.

In IPL-IV a prefix is used (c) to indicate the degree of reference. In the running text, whenever it is important to distinguish symbols, cells, and referents, we will use the same notation. Thus if W0 is a cell, we have:

W0	the cell whose name is "W0".
0W0	"W0" (c = 0).
1W0	the symbol in W0 (c = 1).
2W0	the symbol in the cell whose name is in W0 - i.e., whose name is 1W0 (c = 2).
3W0	the symbol in the cell named 2W0. (c = 3).
7W0	the thing referred to by 1W0 - i.e., as if the symbol in W0 had been written down on this page (c = 7).

There is a class of symbols called variables - e.g., the V's. Structurally a variable is a storage cell. The variable has the value of referent of the symbol that is in it. Thus, if we have a variable process, it must always be executed with a 71 prefix. The reason for this way of defining variables, rather than more directly, is to allow save and discard operations on variables.

Attributes in IPL-IV are active - that is, if the attribute is performed as a process it does the normal process according to the nature of the value. Thus if the value of an attribute is data, performing the attribute is the same as finding the value of the attribute; if the value of the attribute is a process, performing the attribute performs the process (after discarding the operand in HO). One consequence of this is that if a process is needed as data, G16 must be used explicitly.

Goals are pervasively used in GPS. All processes on goals assume them to be in WO, rather than in HO. This includes all attributes on goals, which automatically take the symbol to be in WO, and therefore don't require special loading of the operand into HO. Likewise, all processes that produce goals leave these in WO (with a save) rather than in HO.

There are three special classes of processes that have standardized conventions:

TEST — A process whose sole output is the setting of J. J is set + if the condition as stated in the test is fulfilled; otherwise it is set -. All operands are consumed.

FIND — A process whose output is a symbol to be found, and where it is uncertain whether the symbol even exits. Thus J is set + if the symbol is found, - if it is not. All operands are consumed. If the symbol is found, it is placed in HO, if not, there is no output.

GENERATOR — A process which applies another process to a sequence of operands. This process is always the top operand - i.e., (0). All operands are consumed. Further, in contrast with the usual assumptions about the hierarchy of routines, the generator guarantees that all occurences of the iterated process (called the operation part of the generator) and the higher routine that uses the generator are all in the same context - i.e., as if the generator was not

interposed. The X's are an exception to this, but HO is not. However, nothing can be communicated upwards through the generator. Communication from the operation to the generator is possible via H6 (1H6 = K). Each generator sets K = at the start, and interrogates it after each occurence of the operation. If K is no longer =, a communication has occurred. Generally, the only communication is to stop the generator, and by convention any + will do this.

IV. Detailed description of GPS-I

Vocabulary. GPS uses four regions: Rxxx, Uxxx, Vxxx, and Wxxx. A subject matter system can use GPS only of it doesn't use any of these regions for its own vocabulary. On the other hand, GPS agrees to use only these four regions.

To give an overview of the vocabulary, we list below the major classes of words.

Rxxx GPS processes.

- R00x housekeeping for goals;
- R01x goal types (variables);
- R02x goal formation;
- R03x association of entities together;
- R04x housekeeping for goal components;
- R05x methods;
- R06x evaluations (values for goal types);
- R07x housekeeping miscellaneous;
- R08x tests on goals;
- R09x housekeeping for matching processes;
- R10x matching processes;
- R11x input;
- R12x try-operator processes;
- R13x subject matter set-ups;
- R14x executives.

Uxxx GPS attributes.

UOOx goal attributes;

UOlx goal component attributes;

UO2x total expression, difference, and operator attributes;

UO3x goal attributes.

Vxxx GPS variable routines for subject matter.

VOlx miscellaneous;

VO2x housekeeping: fetch, copy, erase, and file.

Wxxx GPS data: cells, lists, numbers, etc.

The definitions of the W's are given below, since there are only a few of them. Their explanations are to be found mostly via the processes that use them.

WO 1WO = current goal.

W1 1W1 = substitution component for U1.

W2 1W2 = substitution component for U2.

When substituting values for variables, it is necessary to consider a larger set of expressions than the one actually being worked on. W1 and W2 allow separate specification of the component through which substitutions must be made, and the component currently being examined - i.e., U1 and U2, respectively.

W7 7W7 = location program for modification of U12 (see U12, and R99).

W9 W9 = order of generation for total expressions.

All new total expressions are tagged with this number (it then being increased by one) in order to help interpret the data.

W10 W10 = order of generation for goals.

W11 W11 = list of V's.

GPS entities. In the descriptive part of this paper reference was made to a number of entities other than processes that populate GPS: goals, goal components, expressions, differences, operators. In general, these are "pure" symbols, in the

sense that there are no referents for them. Thus, for example, GPS deals with goals. There is a symbol that sits in cell WO, and through it GPS can find out everything that it needs to know about the goal - what other goals it is tied to, what type of goal it is, etc. This symbol acts in every way like a name of a goal - except that there is no "thing" to which the name refers. In all of these cases we tend to consider the symbol to <u>be</u> the goal - i.e., writing 1WO for the goal and not 7WO.

These entities are characterized by the kinds of information that can be obtained from them - that is, by the attributes. Hence the appropriate way to define the various attributes is along with the discussion of the entity to which they belong.

<u>Goals</u>. A goal is a symbol that associates together information necessary to guide behavior. Goals are of various types (UO), and are composed of a number of goal components. Access to the components is by name - e.g., U1, U2, U3 or U4. The type is really a form, and the components are values. For example, type R10 is "find a way from U1 to U2." Particular goals are made by giving particular values to the components U1 and U2. This form is expressed behaviorally in the conduct of the methods that the goal type selects to work on the goal. A goal will have only those components indicated by its type.

A goal has the following attributes:

UO the goal type (executes it);

U1 goal component for expression #1;

U2 goal component for expression #2;

U3 goal component for operator;

U4 goal component for difference;

U5 list of subgoals generated for this goal, recent ones first;

U6 supergoal (goes in WO with save);

U7 list of forward coordinate goals;

U8 list of backward coordinate goals;

U9 final subgoal that achieved this goal;

U30 order of generation;

U31 total expression produced in connection with this goal (used only with the goal most immediately connected with production);

U32 list of methods tried, with repetition and recent ones first.

U5 - U9 represent the direct relationships currently possible between goals. A goal may generate subgoals (several) to which it then becomes the supergoal (unique). Some of these subgoals, which are all coordinate as to level, form a series of steps which lead to the accomplishment of the supergoal. Along one of these series we can associate forward and backward (neither unique). If any of these series is completely successful, the final subgoal is tagged U9 (unique) which helps to tag the successful path created through the network of subgoals.

Two types of goals - R13 and R14 - are really sets of goals, one an "or" set and the other an "and" set. The goals that comprise a set are on the list of which the goal is the name- i.e., this list is not gotten through an attribute, but directly through the goal symbol.

Goal components. A goal component is a symbol that determines - and gives information about - a specific entity that enters into a goal. At the moment there are components only for expressions, operators, and differences, but others could be added. For the operators and differences the components are the entities themselves, since no other specification is needed. (It may become necessary if the operator or difference has free variables in it.) The expression component is more complicated, having several parts accessible via attributes. All three types of components can be sets as well as singles. That is, a component for an expression could turn out to be a component for a set of expressions. Limitless hierarchy

is allowed, although it hardly seems required (sets of sets of singles do occur, however). U18 is a test for a set component. If it is, the set is listed on the list of which the component is the name. The order on this list may be very important, but is determined by context, and is not symbolized anywhere. U18 is the only attribute which is common across all types of components. The other attributes will be discussed with the particular entity to which they apply.

Expressions and total expressions. All objects are treated by GPS via their representations in expressions, which are list structures satisfying a number of requirements as stated by the V's. An expression is made up of subexpressions. None of the subexpressions is considered describable (in the technical sense of IPL-IV) by GPS, although it may be for the subject matter routines. A complete expression that is not a subexpression of any other expression is considered a describable object, and is called the total expression (TEX). All subexpressions are simply called expresssions (EX) including the main expression (the one with no superordinate expression) considered as a non-describable entity.

The components determine expressions, and to specify one the total expression is given plus a location program (U12)which, if applied to the name of total expression as operand (0), will produce the location of the desired expression for (0). We use this "coordinate" method of determining expressions, rather than direct naming, both because it is invariant under copies of the total expression, and because examination of the location program as data can reveal something about the expression located. In consonace with general IPL- IV procedures for getting around on list structures, what is found is the location of the expression rather than the expression itself.

Total expressions also admit free variables, - i.e., ones for which substitutions can be made - and during the course of manipulations various variables get specified. In order to preserve the integrity of the original total expression,

no substitutions are made in it, but in copies of it (U13). Further, a list is kept of the variables that have been specified along with copies of the expressions that were used for their values (U19). Thus the same total expression may appear in many components with different variable specification lists. (Orderwise, it is necessary to make the substitutions first before applying the location program or it may "run off the edge" of the expression.)

Summarizing, a component for an expression has the following attributes:

U11 name of total expression (total expression may still be off on drum);

U12 location program (execute it on (0) = TEX, with output (0) = location of expression);

If there is no program at U12, considered as a data attribute, then applying U12 gets the main expression.

U13 copy of total expression U11.

U19 list of variables followed by the expressions that are to be substituted for them (in pairs).

Total expressions also admit a number of attributes that GPS has access to:

U24 list of goals using entity (here, TEX);

U25 goal that produced TEX (1WO) (unique);

U26 TEX from which this TEX is derived (unique);

U27 list of TEX's derived from this one;

U29 order of generation.

Operators. The only thing associated with an operator is:

U20 try operator (execute).

This is discussed more fully under the processes R120 - R129. Operators also admit U24.

Differences. Differences currently have only two things that comprise them:

U22 test if no difference (execute);

U23 generate related operators (execute).

CIP #7

The "test if no difference" is discussed more fully in the LOGIC section of this paper. U23 is supposed to provide operators that reduce or in some way effect the difference. The difference is really just a symbol for associating a remedy (U23) with a perceivable discrepancy (U22). Differences also admit U24, of course.

V's - subject matter variables. At the moment there are 18 V's which mediate the effect of subject matter on GPS. They are all processes. This emphasizes that GPS is a supervisory program, and never interprets its subject matter - i.e., discriminates among data describing the subject matter. The V's form a motley collection. Most of them are trival, just reflecting the fact that GPS doesn't even know the formats of the expression it is using. Only V0, V1, and V5 carry any weight, problem solving wise. This is a bit deceptive, of course, since V1 really implies a whole set of differences in some specified (and consequential) order, and each of the differences imply a test which is heavily subject matter dependent and a whole set of operators, which again are heavily subject matter dependent. The essential trick in making GPS subject matter "independent," was to find a way for making these various subject matter entities call each other out via some simple associations.

Below is a list of the V's, and some discussion of them:

V0 generates the locations of pairs of corresponding subexpressions.

Input: (0) = operation
(1) = location of expression (#2);
(2) = location of expression (#1).

Output (to operation):
(0) = location of subexpression (#2);
(1) = location of subexpression (#1).

This is the routine referred to earlier that matches two expression according to structure, and allows the determination of any differences there might be between corresponding subexpressions.

V1 generates differences in the order of investigation.

Input: (0) = operation.

Output (to operation):
(0) = difference.

This is the process used in conjunction with V0 to do a match. V0 scans the expressions and V1 determines which differences will be noticed and in what order.

V2 Substitute expression (2) for variable (1) throughout the total expression (0). Discard all operands.

V3 Extract element located by (0) and place it for (0).

The extract operation is a way of removing an expression from the total expression. It is the reverse operation of a substitution. The exact mechanics may depend on subject matter. It guarantees, however, that the total expression of which this expression was a part is still legitimate - so it can be processed further - and that the expression is completely isolated from it - so that no processing of the one affects the other - e.g., an erase of the total expression.

V4 generate all variables of total expression (1).

Input: (0) = operation;
(1) = name of total expression.

Output (to operation):
(0) = variable (not location of variable).

V5 adjustment of forms to expression.

Input: (0) = location of expression.

Output: (0) = location of subexpression.

In some cases where a form is being used for an operator, there may be some final alignment of just which part of an expression the form should correspond to. V5 is a process that applies to the expression which locates the appropriate subexpression. The actual example that has arisen involves the form of the operand fitting to the positive part of a logic expression independent of how many negation signs it has.

V6 find main expression of total expression (0).

V7

V8 test if (0) is a free variable.

V9 generate the free variables in use for the subject matter.

This generator must be able to generate free variables indefinitely. It is needed to have a sufficient supply of free variables to avoid spurious identification because the names of two independent free variables happen to coincide.

V10 move in expression (0) from drum.

V11 copy expression (0), replace (0) with it.

V12 file expression (0) on drum, discard (0).

V13 erase expression (0), discard (0).

V14

V15 move in total expression (0) from drum.

V16 copy total expression (0), replace (0) with it.

V17 file total expression (0) on drum, discard (0).

V18 erase total expression (0), discard (0).

V19

Processes. Very little can be said generally about the processes. We list them with comment associated with the various groups of processes, and with individuals.

R0 - R9 Housekeeping for goals.

R0 Create a new symbol in WO, save 1WO.

R1 Replace (0) by 1WO, save (0), discard 1WO.

R2 Replace 1WO by (0), save 1WO, discard (0).

R3 Replace (0) by P.1WO, discard P.1WO.

R4 Bring in goal 1WO from drum.

R5 File goal 1WO on drum, discard 1WO.

R6 Bring in goal P.1WO from drum.

R7 File goal P.1WO on drum, save name - i.e., leave symbols in WO alone.

R8 File goal P.1WO on drum, delete name - i.e., discard P.1WO.

R9 Erase goal 1WO, discard 1WO.

R10 - R19 Goal types. These are variables whose values are the evaluation processes that select methods for the goal type. Their input is always the particular goal of their own type (assumed) sitting in WO. The output varies, but always sets J: + if the goal is acheived and - if not. Thus all goal types are stated in a form which allows this dichotomy.

R10 Find a way from U1 to U2.

Output: (0) = name of list of R12 goals.

R11 Reduce difference U4 between U1 and U2.

Output: (0) = component with new total expression which is modification of U1 (only if J +).

R12 Apply operator U3 to U1.

Output: (0) = component with new total expression produced from U1 (only if J +).

Note that all operators are assumed to produce new expressions and not just modify the old one. Thus U1 is unchanged.

R13 Accomplish at least one of the goals listed.

R14 Accomplish all of the goals listed.

In both R13 and R14 the goal symbol is the name of a list of goals.

R20 - R29 Formation of goals. These are simply housekeeping processes which take as input components and form new goals using them. They make copies of the components, rather than actually incorporating them into the new goals, since these components are inevitably being used in other goals. They also record the order of generation of the goal. Since the outputs are goals, they are placed in WO with a save. They discard all operands.

R20 Form a R10 goal.

Input: (0) = name of U2 component;
(1) = name of U1 component.

R21 Form a R11 goal with no difference component (U4).

Input: (0) = name of U2 component;
(1) = name of U1 component.

R22 Form a R12 goal.

Input: (0) = name of U3 component;
(1) = name of U1 component.

R23 Form a R13 goal with no goals on list.

R24 Form a R14 goal with no goals on list.

R30 - R39 Association of entities. These processes simply set up pairs of attributes such as subgoal - supergoal. They rarely discard any of the operands.

R30 Link subgoal 1W0 with supergoal P.1W0.

R31 Link forward coordinate goal 1W0 with backward goal P.1W0.

R32 Link total expression of component (0) with producing goal 1W0.

R33 Link total expression (0) with derived total expression (1), discard (0).

R34 Link object (0) with using goal 1W0.

R40 - R49 Housekeeping for components. These processes carry the bulk of the information shuffling - the erasing, copying, and substituting of components. Many of them are recursive, since they must be prepared for components that are sets of other components.

R41 Copy goal component (0), replace (0) with it.

Most of the component can be copied with a G53, which copies the description list as well as the main line. In addition, it is necessary to explicitly copy the location program (U12), the variable-specs list including copies of any expressions mentioned on it (U19), and the copy of the total expression (U13). Further, it is necessary to recurse if the component is a set.

R42 Erase goal element (0), discard (0).

R43 Erase all copies of total expressions (U13) throughout component (0), discard (0).

Simply uses V18 to erase the total expression, and G15 to erase the attribute U13 throughout (0) by means of recursion.

R44 Erase all U13 attributes throughout component (0), discard (0).

R44 differs from R43 in not erasing the total expression. It is needed because multiple reference to a set of copies often gets set up in the course of a process.

R45 Produce copies of all total expressions (at U13) throughout component (0), discard (0). Make substitutions of specified variables (U19 lists).

Again, this is a recursive process. The only complication arises because in a compound set of components the substitutions to be made may be spread out in a hierarchy of U19 lists. R45 dodges this problem by using R46.

R46 Produce copies of all total expression (at U13) throughout component (1), making substitutions from variable-spec list (0) plus any local U19 lists as might exist. Discard (0), (1).

The difference between R46 and R45 lies in the specification list (0). This allows accumulation of a single long list (via G58) as R46 descends the hierarchy of the component picking up U19 lists. Any inconsistency in what to substitute for given variables is resolved on a first-come-first-serve basis (new U19 lists get added at the end of the existing list).

R47 Substitute from variable-spec list (0) in total expression (1), discard (0), (1).

R48 Substitute expression (2) for variable (1) throughout component (0) (copies at U13). Update the U19 list, discard all.

This is the process for making new substitutions in a component. It not only must be recursive in order to handle sets of components, but must seek out the first U19 list it finds to update, and then not modify any of the U19 lists lower in the hierarchy of the component. It does this by setting J + at the beginning, and recursing with a subroutine (*10) which senses this signal, and changes it to - when the first U19 list is found. It also sets up a U19 list if it arrives at a total expression without finding one.

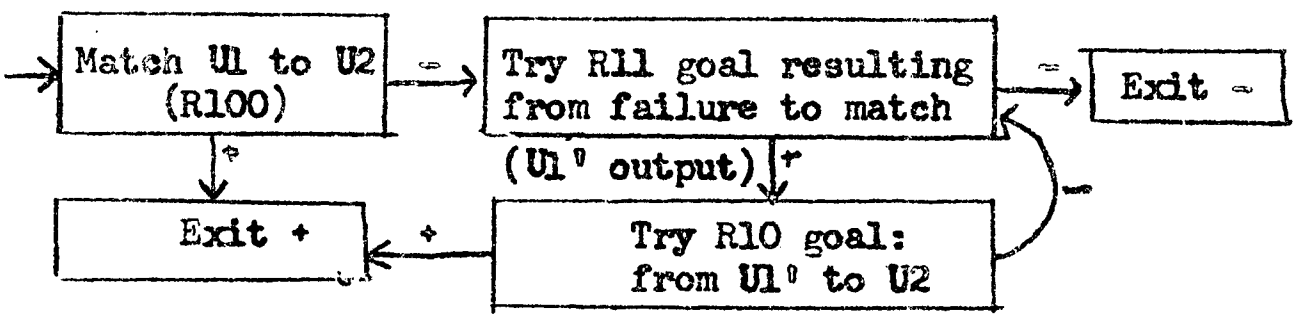
R49 Change the free variables of component (0) to be disjoint from the free variables of component (1), discard (0), (1).

The routine for R49 is rather long and complicated. The problem is to avoid coincidental use of identical free variables in components that are to be compared. To accomplish this R49 lists all the free variables in (1), and then all the free variables in (0), dividing these into the ones that are common - and need to be changed - and those that are not. To get these lists, it must construct a generator (*30) for finding all the variables in a component, and must weed out multiple occurrences. Finally, it proceeds to make the changes, utilizing a generator of free variables provided by the subject matter (V9), and checking against the list of variables already in use in both expressions.

R50 - R59 **Methods.** The real content of the problem solving that GPS can perform is embodied in the methods (plus, perhaps, the evaluations, which serve only a discriminative function among possibilities defined by the methods). They all take as input a goal in W0, from which they can get the inputs they really need. They all set J + or - corresponding to achievement of the goal or not. Thus these signals can be communicated right through the evaluation routines as the outputs to these processes. Their other outputs are varied. In general, they expect the information they need to be associated with the goal, and are not coded for exceptions.

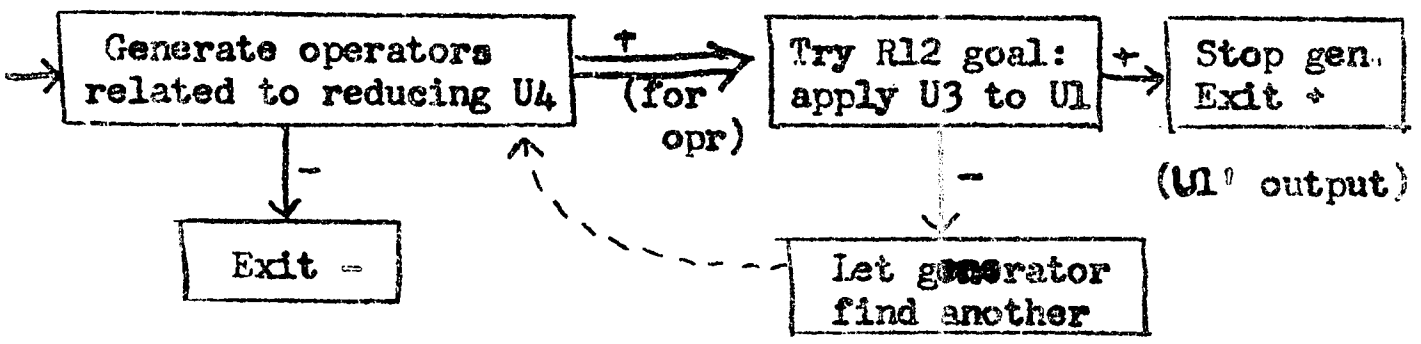
R50 Method 1 for R10: work forward by successive reduction of differences between the initial expression and the final one.

Flow diagram for R50:



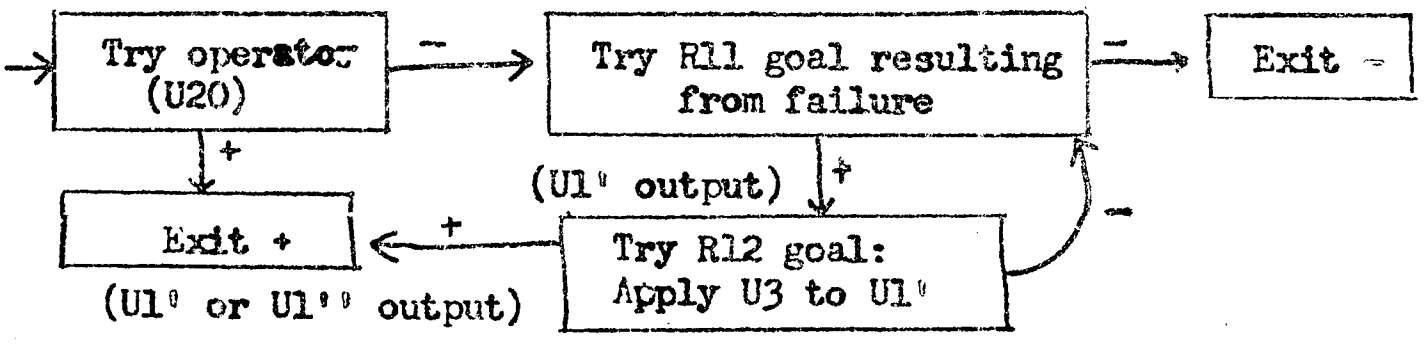
R51 Method 1 for R11: find some operators that reduce the difference by modifying U1, and try them.

Flow diagram for R51:



R52 Method 1 for R12: try to apply the operator; if it fails, try to reduce the difference between the expression and the expression U1 should be like for the operator to apply. R52 usable only with operators which provide a difference type of goal if they fail (other types than R11 may be possible.)

Flow diagram for R52:



R60 - R69 Evaluations. So far, the evaluations are very primitive, partially due to the existence of only a single method for each goal type, and partially due to the lack of experience with the program. The one additional function performed by all evaluations is to keep the number of goals in the core storage to a minimum by filing away the supergoal before executing the method, and bringing it back in at the end. Thus, WO seems like a push down list full of goals; actually only the names are in core storage, except for the top one. Evaluations are all values for goal type variables.

R60 Select R50 (value of R10).

R61 Select R51 if it hasn't been selected before (value of R11).

R62 Select R52 (value of R12).

R63 Select R51 if it hasn't been selected before, and if there has been progress (R89) over the nearest supergoal of type R11 (value of R11).

R63 climbs up the goal network from supergoal to supergoal until it finds an R11 type goal. It then tests for progress via R89, which is a variable test. The need for testing if R51 has been tried, both in R63 and R61, stems from the feedback to the R11 goal in methods R50 and R52. If no R11 type goal is found R63 proceeds as if progress had been made.

R70 - R79 Miscellaneous.

R70 Assign (1) to be value of attribute (0) of 1WO, discard.

R71 Record order of generation for goal 1WO.

R72 Record order of generation for total expression (0), and make into component.

Both R71 and R72 are just convenient abbreviations to save space, since they occur in several routines.

R80 - R89 Tests on goals. Comparisons between goals are the building blocks of the evaluation routines. So far, we only have ideas for R11 goals.

R80 Test if R11 goal 1WO is equivalent to R11 goal P.1WO.

R81 Test if R11 goal 1WO is greater than R11 goal P.1WO.

R82 Test if R11 goal 1WO is less than R11 goal P.1WO.

R80, R81, and R82 allow access to an ordering on the differences provided by the subject matter - and affecting very much the course of problem solution. The ordering of the differences is combined with an ordering on the hierarchy of expressions - any difference detected for an expression is more serious (greater) than any difference detected for its subexpressions.

R88 Test if method (0) has already been tried on goal 1WO, discard.

R89 Test if R11 goal P.1WO is progress over R11 goal 1WO (variable.)

R90 - R99 Housekeeping for matching processes. Matching processes are fairly central for the kind of problem solving GPS does. Their outputs, instead of being simply "succeed" or "fail", are R11 goals which give information about what occurred in the matching process. This is also necessary in a successful match, since many variables may have been specified to achieve it. As a result, several W*s are needed to hold information relevant to a match. Further, whenever a difference is detected, some processing is needed to set up the difference goal.

R90 Set up for match, discard (0), (1).

Input: (0) = U2 component = substitution component (W2);
(1) = U1 component = substitution component (W1).

R90 is simply a special case of R91 for convenience of input.

R91 Set up for match, discard all.

Input: (0) = substitution component (W2);
(1) = substitution component (W1);
(2) = U2 component;
(3) = U1 component.

R91 saves the current values of W1, W2, and W7; produces copies of the total expression (at U13); applies R49 to take care of free variables, and creates the R11 goal through which the match routine will communicate its result. (R90, and R91 are used prior to the match, and not within it.)

R92 Set up W*s.

R93 Clean up W*s.

R99 Fix up components of R11 type goal for difference, discard (0).

Input: (0) = difference (U4).

The main task at the moment of R99 is to produce the correct location programs for U1 and U2 of the goal.

R100 - R109 Match processes. A match is a process of comparison between two complex objects. It assumes they can be decomposed into components, and that there is a way of putting components into correspondence. The input to a match is an R11 type goal (1WO), with no difference (U4). Its output is to set J + if the match is successful - i.e.,

the two expressions are identical; and set J = if there is a difference. In either case the R11 goal contains the information on any changes made in the expression, and gives the difference if it exists.

In GPS, matches are unaware of the differences they are detecting, and consequently, of the changes that might be taking place in the expressions. The function of making an immediate change in the expressions in order to make them match - such as specify some free variables - is delegated to the differences.

R100 Match U1 to U2. Assumes both U1 and U2 are single.

R101 Match U1 to U2. Assumes both U1 and U2 are pairs (components of two singles). Matches in the order in which the pairs are listed - e.g., first single of U1 to first single of U2.

R102 Match U1 to U2. Assumes both U1 and U2 are pairs. Finds best order of match, with preference to given order if possible.

R102 tries R101; then R101 with order of singles in U2 reversed, if R101 hasn't produced a perfect match. It will choose the reversed match if it is perfect, or if it has a match on one of the singles, whereas the original R101 failed to match either. No tests are made on the size of the differences that are found, only on whether 0, 1 or 2 matches occurred on the singles.

R110 - R119 Input processes. Internally, GPS keeps things organized almost exclusively in goals and components. In inputting it should be unnecessary to add all the additional symbols and attributes necessary to make (internally) legitimate components and goals. Instead we adopt a punctuation scheme where the punctuation marks are processes necessary to carry out conversion to the appropriate internal form.

We assume that the kind of entity being input is given by context - i.e., that some higher processes will be expecting the input lists, and will know, e.g., that they are to be operator components. Therefore, processes are provided for use by the higher routine to make the conversion (R116 and R117 so far). However, uncertainty still remains about whether components are sets or singles, and if sets, just how many of the input total expressions should be made into one set. Thus for punctuation we use R111, R112, R113, and R114 to stand for one to four singles in a component respectively. Finally, R115 stands for a list of singles to be made into a component.

Specifically, the input conventions are:

1. The input of an entity is to be a list containing the names of the total expressions making up that entity;

2. Components are given by listing the names of the total expression in the same order in which they should occur in the component (if a set), followed by a symbol giving the number of total expressions in the component;

3. Names of total expression are written with a 10 prefix (IPL-IV code).

R110 Useful subroutine for R112 - R115.

R111 Construct single component for total expression (0), replace (0). Component specified main expression and no U19 list.

R112 Construct a set component of total expressions (0), and (1), replace (0), discard (1). Singles like R111.

R113 R112, but three total expressions.

R114 R112, but four total expressions.

R115 R112, but (0) = list of total expressions.

R116 Convert input expression (0) to form component, discard (0).

R117 Convert input expression (0) to R10 goal, discard (0) (goal is 1W0, with save).

R120 - R129 Try-operator processes. In general it cannot be assumed that an operator is applicable to a given expression; and the operator, when tried, must be prepared to produce differences as output, if unsuccessful, as well as new total expressions if successful. This responsibility is delegated to the operators individually - each may do it in its own way.

However, one general class of operators is specified by giving a pair of forms - the condition form, and the product form. The operator is applicable if the condition form can be fitted to the expression. The product form gives the result, using the identifications that occurred in fitting the condition form. Thus the condition form really serves two functions. The three processes coded in this section so far (R120, R121, R122) are the common parts of this operation of using forms under three slightly different conditions. They allow us to specify the operator just by giving a form component.

R120, R121, R122 have common input and output specs:

Input: (0) = form component of operator;
(1) = component of expression to be operated on.

Output: if successful, set J+ and (0) = component for newly produced total expression;

if unsuccessful, set J - and 1W0 = R11 goal;

discard inputs.

The three processes differ in their assumption about the input components, and how they produce their product. Note that in all cases a new component with new total expression is generated, even it the operator is viewed a modifying the input - e.g., commutation.

R120 Try-operator. Assume input components single; adjust form to expression; replacement type operator - i.e., modified subexpressions in total expression.

R121 Try-operator. Assume input components single; expression must be main; produces new expression - i.e., not just a modified copy.

R122 Try-operator. Assume input components are pairs; both singles of operand must be main; produces a single new total expression.

R130 - R139 Subject matter set-ups. One of the virtues of GPS is its ability to change subject matters easily. This is a necessary condition for it to be a general problem solving routine. Further, various techniques that seem very important for advanced problem solving - like planning - require frequent shift of subject matter during work on a single problem. These subject matters are all related, being various approximations to each other, yet from a technical standpoint they are different - i.e., GPS must be adapted in moving from one to the other.

All subject matter dependent parts of GPS have been concentrated in the V's. All that is needed to change subject matters is to install new values for the V's appropriate to the new subject matter. By usual convention we make the V's sage cells - i.e., in changing the current values of the V's must not be clobbered.

R130 Set up for change in subject matter.

At the moment, simply saving all the values of current V's.

R131 Clean up and restore previous subject matter.

At the moment, simply discarding all current values of the V's.

R132 Set up GPS for W/R logic.

R133 Set up GPS for operator logic.

The only difference between the two forms of logic indicated in R132 and R133 is that in "operator logic" one can eliminate double negations, and thus work with a system in which sign change is the relevant operation on negation signs.

R140 - R149 Executives. These are the processes that are prepared to accept certain information from the outside and initiate goals to be accomplished.

V. LOGIC for GPS

In order to use GPS for symbolic logic some special processes must be defined to fit the V's, and some entities set up - differences, operators, etc. All of this can be done within the framework of LOGIC, the regular technical vocabulary for symbolic logic in IPL-IV.

The following assignments are made:

Axxx attributes - standard.

Cxxx logical constants and variables - standard.

Dxxx differences - special for GPS.

Lxxx lists - standard, but some redefined:

L0 - L9 - standard;

L10 - L19 - miscellaneous for GPS;

L20 - L49 - lists of operators related to D0 - D29;

L50 - L99 - lists of forms components for operators Q0 - Q49.

Mxxx problems in operator logic - special for GPS.

Pxxx logic processes - standard, but some redefined:

P0 - P49 - standard set of logic basic processes;

P50 - P69 - miscellaneous for GPS;

P70 - P89 - test if no differences - for D's;

P90 - P119 -

P120 - P149 - generators of related operators - for D's;

P150 - P199 - try operators - for Q's.

Qxxx operators - special for GPS.

Yxxx data cells - standard, although probably not needed.

All but a few of the special P's are trivial. Many are one word processes: adding a symbol of information, such as the name of a list of related operators

or of an operator form component, and referring the job to a standard process such as G91 (generator of a list), or R120 (try-operator). The ones that have real substantive content are P50, the generator of corresponding pairs of expressions, and the tests of no difference, P70 and on. We give a few comments on these below.

P50 Generate pairs of corresponding major elements of expression located by (1) and expression located by (2).

Output (to operation):
(0) = location of expression of (1) above;
(1) = location of expression of (2) above.

Note that this generates **major** elements - the binary element or variable plus all the nots that apply to it. P50 works down the tree of the hierarchy of the logic expression, always taking the left subelement if possible.

P70 Test if no difference - for D0, set of variables.

Input: (0) = location of expression;
(1) = location of expression.

Output: If J +, no output;
if J -, fills out R11 goal.

P70 first tests if there is any variable in expression located by (0) that is not also in expression located by (1). If there is one, it outputs a difference D10 (add variable). It then tests if there is any variable in expression located by (1) that is not also in expression located by (0); and if it finds one, outputs a difference D11 (delete variable). P70 ignores free variables.

Note that, although the difference we are testing for is D0, the differences that show up in the R11 goal are further differentiated - i.e., D10 or D11, where the direction of the difference is indicated.

P71 Test if no difference - for D1, number of occurrences of variables.

Input and output are like P70; in fact, all tests of no differences must have identical formats. P71 assumes an identical set of variables exists between the two expressions (it, too, ignores free variables). Thus it is logically posterior to P70. P71 takes each variable in turn and counts the number of its occurrences in the two expressions. If it finds a variable that has more occurrences

in the expression located by (0) than in the expression located by (1) it outputs a difference D12 (add occurrences). If the other way around it outputs a D 13 difference (delete occurrences). It gives D12 priority over D13 - any D12 difference will be detected before D13 differences are looked for.

P72 Test if no difference - for D2, negation signs.

Basically, P72 detects a difference in number of negation signs in the two elements. It outputs the difference D2, not differentiating which side has the excess negation signs. However, it does see if a suitable substitution of a free variable will make the negations the same, and if so makes it. It uses R48 for the substitutions, thus assuring that they get made throughout the appropriate sets of total expressions.

P73 Test if no difference - for D3, variables versus expressions.

P73 assumes that the negation signs of the two elements are the same, thus being logically posterior to P72. The difference that it detects (D3) is an inequality of structure - one expression having a variable where the other has an entire expression. It also detects (and still calls D3) two variables which are different. Like P72, it attempts to make substitutions for free variables so as to make the expressions the same.

P74 Test if no difference - for D4, connectives.

P74 assumes two binary elements with the same negations. Thus it is logically posterior to both P72 and P73. It simply notices whether the connectives are the same or not.