Complex Information Processing
Working Paper #227
January 2, 1973

Problem Solving and Rule Induction: A Unified View

by

Herbert A. Simon and Glenn Lea
Carnegie-Mellon University

Discussions in the psychological literature of cognitive processes generally treat separately a category of behavior called "problem solving," on the one hand, and a category called sometimes "concept attainment," sometimes "pattern induction," and sometimes "rule discovery," on the other. We will use the phrase "rule induction" to refer to any of the diverse tasks in the second category. We find this division already in the 1938 edition of Woodworth's Experimental Psychology, where the penultimate chapter is devoted to problem solving behavior, and the final chapter primarily to rule induction. In explanation of this organization, Woodworth comments (1938, p. 746):

> Two chapters will not be too many for the large topic of thinking, and we may make the division according to the historical sources of two streams of experimentation, which do indeed merge in the more recent work. One stream arose in the study of animal behavior and went on to human problem solving; the other started with human thinking of the more verbal sort.

Far from merging, the two streams are still treated as quite distinct in more recent works. For example, in his 1968 Annual Review survey of artici-cial intelligence studies and their relevance to psychology, Earl Hunt devotes separate sections to "deductive problem solving" and "inductive problem solving," his categories corresponding closely to those introduced above. Similar categories appear in the principal contemporary textbooks.

This dichotomization cannot be regarded as satisfactory, for it fragments theories of thinking into subtheories with no apparent relation between them. In proposing information processes to account for problem solving, the theorist then assumes no responsibility for the relevance of these processes to concept attainment or other rule induction tasks, and vice versa. It is of course possible that these two kinds of thinking activity are entirely separate and independent, but possibility is not plausibility. It would be much better if we could show just how they are related; or, if they are not related, if we could provide a common framework within which the two classes of activities could be viewed.

Hunt's (1968) dichotomy of "deductive" and "inductive" will not do, for it is easy to show that from a logical standpoint the processes involved in problem solving are inductive, not deductive. Hunt may have been misled by the fact that the earliest artificial intelligence systems for problem solving (e.g., the Logic Theorist) dealt with the task environment of theorem proving. Clearly, the proof of a theorem in a formal mathematical or logical system is a deductive object; that is to say, the theorem stands in a deductive relation to its premises. But the problem solving task is to <u>discover</u> this deduction, this proof; and the discovery process, which is the problem solving process, is wholly inductive in nature. Hence, both a theory of problem solving and a theory of rule induction must explain inductive processes--a further reason for believing that these theories should have something in common.

Recent developments in the theory of problem solving (Newell, 1968; Newell & Simon, 1972; Simon, 1972) give us a clue as to how to go about building a common theory that will embrace both problem solving and concept attainment. It is the aim of this paper to outline such a theory. In order to attach the discussion firmly to empirical data, certain specific tasks, belonging to each class of behavior, that have been much studied in the laboratory will be examined as typical of their respective classes. For problem solving, we will pay special attention to two tasks analysed at length in Newell & Simon (1972): cryptarithmetic, and discovering proofs for theorems in logic. For rule induction, we will use as examples the standard concept attainment paradigms (Bruner, Goodnow, & Austin, 1956; Hunt, 1962; Gregg & Simon, 1967), extrapolation of serial patterns (Feldman, Tonge & Kanter, 1963; Simon & Kotovsky, 1963; Simon, 1972), and induction of the rules of grammar (Solomonoff, 1959; Klein & Kuppin, 1970; Siklossy, 1972).

Our undertaking is a little more ambitious than we have indicated. For, not only have distinct bodies of theory grown up to deal with problem solving and rule induction, respectively, but there has been relatively little unity in theorizing across the whole of the latter domain. In particular, theoretical treatments of concept attainment do not discuss extrapolation of patterned sequences as an analogous activity, and theories of sequence extrapolation do not encompass the standard experimental paradigms for studying concept attainment. Hence, we will aim at a unified treatment of the whole range of things we have here been calling "rule induction," and a unification of these, in turn, with the activities called "problem solving."

We will begin by outlining the developments in the theory of problem solving to which we referred above, and then use these developments to construct the broader theory.

Problem Solving

The general structure of the information processing theory of problem solving is embedded in four broad propositions (Newell & Simon, 1972, pp. 788-789):

1. A few, and only a few, gross characteristics of the human information processing system (IPS) are invariant over task and problem solver.

2. These characteristics are sufficient to determine that a task environment is represented (in the IPS) as a problem space, and that problem solving takes place in a problem space.

3. The structure of the task environment determines the possible structures of the problem space.

4. The structure of the problem space determines the possible programs that can be used for problem solving.

Thus, knowledge of the task environment permits us to set broad bounds for the behavior of an appropriately motivated problem solver placed in that environment. Knowing the problem space he adopts for his attempts at finding a solution greatly narrows the bounds. The specific behaviors that occur within these narrower bounds flow from the interactions between the task and the particular problem solving program he employs.

A problem space is a set of points, each of which represents a <u>knowledge state</u>. A knowledge state is the set of things the problem solver knows at a particular point in his search for a solution. For example, at a certain point in his attempt to solve the cryptarithmetic problem, DONALD+GERALD=ROBERT, the subject may know that the number 5 must be assigned to the letter D, the number 0 to T, and the number 9 to E; and may know also that R is odd and greater than 5.

Insert Figure 1 about here

Problem solving activity can be described as a search through the space, or maze, of knowledge states until a state is reached that provides the solution to the problem. Thus, in the cryptarithmetic problem, the solution state is one in which each letter has been assigned a digit, and in which it has been verified that these assignments provide a correct translation of the addition problem. In the cryptarithmetic task the problem solver moves from one state to another by inferences and by visual searches of the problem display. For example, knowing that E=9 and that R is odd and greater than 5, he may infer that R=7. Or, knowing that E=9, he may discover, by scanning, the E in ROBERT, and replace this by a 9, obtaining: A+A=9 (apart from carries) for the third column from the right.

Similarly, in discovering the proof for a theorem, a problem solver organized like the General Problem Solver (GPS) starts with some initial expressions (premises) and the goal expression (the theorem to be proved), and applies rules of inference to generate new expressions that are derivable from the premises, until an expression is generated that is identical with the desired theorem. In this case, the knowledge states of which the problem

----

Insert Figure 2 about here

----

space is composed are sets of expressions that have been derived along particular inference paths.

The search through such a problem space is guided by the information that becomes available at each successive knowledge state. Given that the problem solver has already visited a certain number of points in the problem space, he can determine the direction in which he will continue to search by two kinds of decisions: selection, from among those already visited, of a particular knowledge state from which to continue his search; and selection of a particular operator (inference rule, or "move") to apply at that point in order to reach a new knowledge state.

Means-ends analysis, used extensively by human subjects in many problem environments, is a particular kind of scheme for making the choice of operator. It is the key selection mechanism incorporated in GPS. For means-ends

----

Insert Figure 3 about here

----

analysis, the information in a particular knowledge state that has already been reached in the problem space is compared with the specification of the solution to discover one or more differences between them. Corresponding to one of these differences, an operator is selected that is known, from previous experience, often to eliminate differences of that kind. The operator is applied to reach a new knowledge state.

In searching for the proof of a theorem, for example, an expression already derived is compared with the theorem to be proved. A difference detected between them cues a rule of inference to be applied. If, for instance, the expression already derived is of the form AB and the theorem of the form BA, then this difference (in the order of the terms) may cue the application of a commutation operator, if one is available among the rules of inference.

We may formalize and generalize this description of problem solving as follows: there is a problem space whose elements are knowledge states; there are one or more generative processes (operators) that take a knowledge state as input and produce a new knowledge state as output; there are one or more test processes for comparing a knowledge state with the specification of the problem state, and for comparing pairs of knowledge states and producing differences between them; there are processes for selecting these generative and test processes on the basis of the information contained in the knowledge states.

The crucial point in this characterization is that information contained in the knowledge state can be used to guide the generation of new knowledge states, so that the search through the problem space can be selective rather than random. This formulation shifts the emphasis from the search for a problem solution to a search for <u>information</u> that will reveal the problem solution.

A trivial example will show that these two searches--for a solution, or for information leading to a solution--can be quite different processes. Suppose that I have mislaid on my desk the piece of paper on which I have written the combination of a safe. If I wish to open the safe, I can proceed in either of two ways: I can search through the space of combinations by turning the dial until I find the correct sequence of settings; or I can search my desk for the missing piece of paper that contains the information about the combination. Depending on how elaborate the combination is, and how much clutter there is on my desk, the one space or the other may be the larger and more difficult to search. In any case, there is no fixed relation between the sizes of the two problem spaces.

We wish now to pursue the idea that characterizing problem solving as information gathering rather than as searching for a problem solution gives us the broad framework we need to deal with the whole range of tasks in which we are interested--that the fundamental processes for information gathering in problem solving tasks are essentially the same as the fundamental processes for information gathering in rule induction tasks.

This formulation will also yield some valuable byproducts for the theory of problem solving in its narrower connotation. For it has often been observed that one important technique of problem solving--particularly crucial for the solution of certain "trick" problems--is to change the problem representation, moving from the space in which the problem is initially formulated into some alternative problem space. Planning processes that have been observed in human problem solving also involve shift from an original problem space to another, more abstract, problem space. Hence, it appears necessary, in the theory of problem solving, to accommodate more than a single problem space, and to characterize the processes that exchange information between these spaces. Viewing problem solving as information gathering facilitates this kind of analysis, and permits us to consider a multiplicity of problem spaces.

Information Gathering in Theorem Proving

Consider the following GPS-like system for discovering proofs for theorems in symbolic logic. The knowledge states are sets of logic expressions that have been derived from the initial premises. Two kinds of information are used to discover the solution: the degree of similarity or difference of the expressions contained in a given knowledge state from the goal expression, and the specific character of the differences between particular expressions in the knowledge state and the goal expression. The first kind of information measures the progress that has been made in reaching a knowledge state--if it contains an expression that is highly similar to the goal expression, then it can be taken as a likely starting point for further search. The second kind of information suggests how a closer approximation to the goal expression can be obtained--the specific differences that are detected suggest specific operators to remove them.

To extract this information from knowledge states requires a process for matching pairs of logic expressions until a pair of subexpressions is detected that are not identical. The output of the match process is a symbol naming the particular difference that has been detected and, possibly, its location in the expressions. The system may contain a number of different types of differences that can be detected, and these differences may be parameterized.

For example, the symbol E(A) might denote that the first of the pairs of expressions being matched contains a symbol not contained in the second, and further, that this extra symbol is "A". Notice that the parameter value does not have to be supplied by the matching process, but is extracted directly from the expressions being matched. Thus, the knowledge state provides information both about types of differences, and about the specific character of the differences belonging to these types. If we denote by E(X) the whole class of differences of this type (extra symbol in first expression), then the specific difference, E(A), is obtained by substitution of the constant A for the variable X. No search through a set of possible substituends is required, for the necessary information is available and explicit in the knowledge state.

<u>Information Gathering in Cryptarithmetic</u>

The information gathering process in solving cryptarithmetic problems could be described in a manner very similar to our description of information gathering in theorem proving. It is more instructive, however, to look at matters in a slightly different way. Let us consider the knowledge states in cryptarithmetic to be made up of two components: the <u>problem display</u> in which digits have replaced those letters to which assignments have already been made; and the <u>list of assignments</u> themselves. The problem solving goal can then be described in two ways: (1) to replace all letters in the display by digits in such a way that the resulting problem in arithmetic is correct; or (2) to complete the list of assignments of digits to letters so that each letter has a distinct digit assigned to it. Of course, both conditions must be satisfied

---

Insert Figure 4 about here

---

to solve the problem, but if appropriate consistency checks are made when the display is modified and when a new assignment is added to the list, reaching either goal will guarantee achievement of the other.

How is information extracted from knowledge states in the course of solving the problem? Whenever sufficient information has been accumulated in any column in the display, one or more new assignments of digits can be inferred from it. For example, in DONALD+GERALD=ROBERT, if D=5 has been assigned, so that the display becomes: 5ONAL5+GERAL5=ROBERT, it can be inferred that the last T is 0, so that T=0 can be added to the list of assignments. The inference is made by a "Process Column" (PC) operator that takes the column of the display (together with information about carries) as input, and produces the assignment as output.

Conversely, whenever a new assignment is added to the list, the display can be changed by substituting the appropriate digit for the corresponding letter wherever the latter occurs in the display. For example, suppose we have the display 5ONAL5+G9RAL5=ROB9RO and the list of assignments: (D=5, T=0, E=9). Suppose we now add to the list the new assignment, R=7. We can now alter the display to read: 5ONAL5+G97AL5=7OB970. Here, the input is an assignment from the list of assignments, the output is a modified display. The modification is made by a "Substitution" operator that searches the columns of the display for instances of the letter in question, and substitutes the digit for it wherever it is found.

Other inferential processes for producing new information may operate internally to the list of assignments or to the display, respectively. As an example of the former, suppose that the list of assignments includes the information: E=9 and R=7v9. Then, if there is a process for examining the consistency of assignments, that process can draw the inference that R=7, and replace R=7v9 on the list by this more precise assignment. Similary, processing column 1 of the problem with the information that D=5, leads both to the inference that T=0, and that a 1 is carried into the second column. The latter piece of information can be entered directly on the display.

The situation can now be redescribed in the following way. We consider *two* problem spaces: a space of assignment *rules* (rules for substituting digits for letters in the display), and a space of *instances* (columns of the display). The goal is to complete the list of rules, so that there will be a distinct assignment rule for each letter. The proposed rules are tested against the instances. Each column of the display, which we are now interpreting as an instance, provides a partial test of the consistency of the rules. The situation so described differs from the usual concept attainment paradigm only in the fact that the instances are not completely independent, but interact through the carries from one column to the next (Figure 5) In every other respect, the task is now a standard concept attainment task. Simply by changing our way of viewing the problem space (or spaces), we have transferred from the category of problem solving to the category of concept attainment, pattern induction, or rule discovery.

Insert Figure 5 about here

From this example, we may surmise that the trademark that distinguishes these two classes of tasks is the presence or absence of more than one distinguishable problem space in which the problem solving activity takes place. If there is only one space, we describe problem solving as a search through that space, made more or less selective and efficient by drawing upon the information that is available in each of the information states that is reached. If there are two spaces, we describe problem solving as a search through one of them (usually, as we shall see, through the space of rules), made more or less selective and efficient by using information available in each space to guide search in the other. By focussing our attention on the processes for utilizing information, we can provide the common framework that we have been seeking for all of these tasks.

Rule Induction

Newell (1968,1973) has proposed a taxonomy of methods that can be used by a general problem solver--methods, that is, that make relatively unspecific demands upon the task environment in order to be applicable. Some items drawn from this taxonomy will clarify the way in which information is obtained and used by a problem solving system, and will enable us to examine in detail the information flows in problem solving and rule induction systems.

Some General Methods

We will be concerned with just three of Newell's methods: the generate-and-test method, the heuristic search method, and the induction (or hypothesis-and-match) method. Within variants of these three methods there will be incorporated, in turn, two other submethods: the matching method, and the means-ends method.

We postulate a problem solving task of discovering an object of a certain kind. In particular, the object that is sought may be a rule, pattern, or concept, but need not be any of these. The simplest solution method, _generate-and-test_, postulates two information processes: a generator of potential solutions, and a test to determine whether an object produced by the generator is in fact a solution. Whatever power and efficiency this method has derives from information built into the generator and test. If, for example, the set of objects the generator can produce is very small, and if this set is guaranteed to contain the solution, the method will be powerful. If the test can reject inadequate solutions rapidly--say, by means of a matching process--then the cost of testing will be relatively small. That is about all that can be, or needs to be, said about the generate-and-test method.

Next, suppose that the generator is not independent of what objects have been generated previously--that information flows back from the test to the generator. This means that the test must provide more information than just the success or failure of the match between objects generated and the specification of the goal object. A common form of the dependency of generator upon test is that the generator produces each new object by modifying an object produced previously in the search. It is this kind of dependency that characterizes the _heuristic search_ method.

We have already remarked on two kinds of information that can be used by the generator in heuristic search. In our present terms these are: first, information to select which of the previously generated objects will be modified to produce the next object; second, information to select which of several available operators will be applied to the object to modify it. If the latter choice depends on detection by the test of specific differences between an object and the goal object, then we speak of using the means-ends method.

Now, suppose the problem solving is to occur in a dual space. In the rule induction tasks, the goal object is a rule in the first of two spaces. The test of whether a given object is that rule involves applying the proposed rule to objects ("instances") in the second space, and then testing whether the application gives a correct result. The phrase "correct result" means either that the rule generates instances in the second space which then are determined by the test to satisfy some criteria;or, that the rule is used to classify instances in the second space that have been produced by another generator, and that the classifications are then determined by the test to be correct. Newell defines the induction method as one in which there are separate generators for rules and instances, and a match process to test whether a rule agrees with the instances.

A moment's reflection will reveal that the induction method, so defined, is really a whole collection of methods. In its most primitive version, there is no feedback of information from test to rule generator; the test simply eliminates rules that have been generated, but does not provide information to help the generator select the next rule. In this case, the method is an "inductive" version of the generate-and-test method, adapted to the dual problem space.

On the other hand, if the generator does not create each rule anew, but produces it by modifying previous rules on the basis of information received from the test of instances, then we have an inductive version of the heuristic search method.

Further, the existence of two problem spaces and two generators, one for rules and one for instances, opens up possibilities for methods that are not available when there is only a single problem space. For example, the instance generator may not be autonomous, but may instead derive information from the rules that have been generated, and the previous tests that have been performed. Thus, each new rule may be generated on the basis of the instances constructed up to that point (heuristic search for rules), while each new instance may be generated on the basis of the rules constructed up to that point (heuristic search for instances). This, in fact, is what was going on in the cryptarithmetic solution method described earlier when we interpret the problem display and the list of assignments as defining two distinct problem spaces.

Let us now make these ideas about information flows more concrete by applying them to standard psychological paradigms for experiments on rule induction. First, we will discuss concept attainment experiments, then extrapolation of patterned sequences, then induction of grammars.

Concept Attainment

In the commonest laboratory form of the concept attainment task, the subject sees a sequence of stimuli that differ along one or more dimensions (e.g., "large blue square," "large green triangle," etc.). Certain of these stimuli are instances of a concept (e.g., "square"), others are not. The subject guesses whether each is an instance, and is told whether he is right or wrong. His task is to induce the concept so that he can classify each successive stimulus correctly. In heuristic search terms, the subject can be considered to be searching through a space of possible concepts for the right one. The information that guides this search, however, is not information about concepts, but information about whether certain stimuli are instances of concepts or not.

Gregg & Simon (1967) have described a whole class of inductive methods for performing the concept attainment task which differ with respect to the amount of information the subject retains as a basis for guiding the concept generator. If no information is fed back, whenever a guess has been wrong, the method selects a concept at random from the set of available concepts. A slightly more efficient generator (which does not, strictly speaking, require feedback from instances other than knowledge of whether the last instance was classified correctly or not) would sample randomly from the set of available concepts, but without replacing those already eliminated. A somewhat more efficient generator would produce a concept consistent with the correct classification of the most recent instance. A still more efficient generator would produce a concept consistent with the classifications of all previous instances. Gregg and Simon argued that which of these methods would be employed by a human subject would depend, at least, on the limits of his short-term memory, and the availability of time to fixate information or of external memory to record it.

The paradigm just described does not incorporate the flow of information from the space of concepts to the generator of instances, since the instances are produced by the experimenter independently of the subject's problem solving processes. The two spaces are linked only through the problem solver's guesses as to the correct classification of the instances as they are produced. In fact, these guesses are irrelevant, since the information is actually provided by the experimenter's reinforcement of each guess as correct or incorrect. The same problem solving methods would work if the experimenter simply classified each instance as corresponding or not corresponding to the concept, without demanding a response from the problem solver. The flow of information is entirely from the instances to the concept generator, and not in the opposite direction.

However (Bruner, Goodnow & Austin, 1956), it is also possible to arrange the concept attainment experiment in such a way that the problem solver himself generates the instances. He may, of course, generate them randomly; but he may also select instances so constructed as to choose between two classes of hypotheses. This information flow from the space of rules to the generator of instances enables solution methods that are more efficient than any that are available with a one-way flow of information. Notice that the criterion for selection of instances is indirect and sophisticated: instances are valuable for solving the problem (finding the correct concept) to the degree that information on their classification imposes new restrictions on the domain of the rule generator.

Extrapolation of Patterned Sequences

Contemporary theories of how human subjects discover the patterns implicit in sequences of letters or numbers and use these patterns to extrapolate the sequences have been reviewed and synthesized by Simon (1972).

In the sequence extrapolation task, the subject is presented with series of symbols followed by one or more blanks (e.g., "ABMCDM_"). His task is to insert the "right" symbols in the blanks--that is, the symbols that continue the pattern he detects in the given sequence. The goal object, then, is a sequence of symbols in which all of the blanks have been replaced "appropriately." But to define what is meant by "appropriately," we must introduce the notions, e.g., of "same" and "next" between pairs of symbols, and perhaps other relations, in order to characterize the pattern as a basis for extrapolating it.[1/] If the problem solving is to be characterized as a search, the

1/ Ernst & Newell (1969) have suggested an ingenious scheme for handling the sequence extrapolation task as a problem solving task--that is, in terms of a single problem space that accommodates both the sequences and the patterns. We will not discuss this scheme here, since an analysis in terms of a dual problem space seems more natural and simpler.

search goes on in the space of patterns and not in the space of extrapolated sequences.

To extrapolate the sequence, ABMCDM..., given as an example above, the problem solver must induce the pattern underlying that sequence: in each period of three letters, the first is <u>next</u> in the English alphabet to the second letter in the previous period; the second letter in each period is next to the first letter in the same period; the third letter in each period is the constant letter 'M'. The period might be described as'N2pN1sS3p' with 'S' for "same" (and 'N' for "next") the numerical postscripts to indicate which symbol is to be updated. The sequence is initialized by supplying the beginning 'A' and the constant 'M'.

Clearly, the sequence itself in the extrapolation task is the counterpart of the instance in the concept attainment task; while the pattern is the counterpart of the concept. What are the flows of information? As in the simplest concept attainment paradigm, the sequence is provided by the experimenter rather than the problem solver. However, in his search for pattern, the problem solver can choose which elements of the sequence he will test for relations at any given moment. If, in the previous example, he is provided with three periods instead of two--ABMCDMEFM...--then, having discovered the second 'M' three symbols beyond the first, he can test whether an 'M' occurs again three symbols later. To this extent, there can be a flow of information from a hypothesized pattern component (the repetition of 'M') to a choice of which instance (which part of the sequence) to examine next.

The flow of information in the opposite direction, from sequence to pattern, is even more critical for the efficiency of the solution method. The problem solver need not generate "all possible hypotheses," but can instead detect simple relations ("same" and "next") between pairs of symbols in the sequence, and then hypothesize patterns constructed from those relations. Although obviously inductive, the process need not involve any considerable amount of search.

Induction of Grammars

As our final example of a rule induction task we consider the induction of a grammar for a language from examples of sentences and non-sentences. This task has received some attention in the artificial intelligence literature (e.g., Solomonoff, 1959; Klein & Kuppin, 1970; Biermann & Feldman, 1971; Siklóssy, 1972). In the grammar induction task, the subject generates a succession of symbol strings that may be sentences in a language possessing a formal grammar. He is then told whether or not each string is a sentence. His task is to induce the rules of the grammar so that he can predict infallibly whether any given string is a sentence. The commercially marketed game, QUERIES 'N THEORIES, provides a version of this task that is readily adapted to the laboratory.

In this problem domain, the examples of sentences and non-sentences constitute the space of instances, while the grammar rules correspond to the space of concepts. In the most common form of the task, the problem solver selects the sentences against which to test his system of rules, hence there is a flow of information from the space of rules to the space of instances, as well as a reverse flow from instances to rules.

Let us illustrate these information flows more concretely. Consider a grammar with two components: a set of base sentences; and a set of replacement rules that allow the construction of a new sentence by replacing certain symbols or sequences of symbols in any sentence where they occur by a new symbol or sequence. A simple example of such grammar is given by:

Base sentence: Y

Replacement rule: Y <-- BY

This grammar has a single base sentence, Y, and a single replacement rule, Y <-- BY. Applying the replacement rule to the base sentence, then to the resulting sentence, and so on, we obtain, as additional sentences of the language, BY, BBY, BBBY, and so on.

Suppose that it was already known, by previous tests, that Y and BY were sentences. Then, by supplying information from the instances to the rule generator, the possible replacement rule Y <-- BY could be constructed directly. Reversing the flow of information, the rule itself can now be used to generate instances of predicted sentences, and the correctness of these can be checked by the "native informant" (the experimenter).

To match the various concept attainment paradigms, the task could be modified, for example, to supply all examples of valid sentences in advance. Or the experimenter could supply examples of sentences and non-sentences, and require the problem solver to classify them. The two classes of tasks are in every way identical with respect to the ways in which information can be made available to the problem solver.

A General Rule Induction Program

We have now explored the formal isomorphisms among three classes of rule induction tasks: concept attainment, extrapolation of patterned sequences, and induction of the rules of a grammar. One way to exhibit these isomorphisms quite formally is to construct a single computer program that is capable of performing the whole range of tasks. We can then compare this program, in turn, with a general problem solving program that operates in a single problem space.

We must be careful as to what we mean by "capable of performing the whole range of tasks." The space of concepts appropriate to the usual concept attainment tasks is different from the space of grammar rules or the space of sequential patterns. For a program to undertake to solve problems in any one of these domains, it will require, in addition to its general mechanisms and organization, common to all the domains, particularized equipment for dealing with the specific domain before it.

The situation here is essentially the same as the situation confronting the General Problem Solver. GPS is a very general organization for performing means-ends analysis, and for guiding search through a space of knowledge states. Before GPS can go to work on any specific problem, it must be provided with a specification of the problem domain: the objects, the definition of the knowledge states, the operators, the differences, and the associations of operators with differences. A General Rule Inducer will need the same kinds of problem specification in order to tackle specific tasks. GRI itself will be an executive program providing a framework within which the specialized subprocesses can operate. Figure 6 uses the informal programming language defined in Newell & Simon (1972, pp. 38-51) to describe the program of GRI.

Insert Figure 6 about here

In Figure 7, we show the flows of information in GRI. Some of these (shown by broken lines) are "optional," in the sense that variants of the task can be devised, as we have already noted, that include or exclude them. Thus, in the Bruner-Goodnow-Austin paradigm for the concept attainment task, information from the current hypothesis is an input to the instance generator, while in the more common laboratory paradigm for this task, the instances are generated by the experimenter without using this information.

Insert Figure 7 about here

Table 1 shows how the major processes in GRI are to be interpreted in the context of the specific tasks: concept attainment, sequence extrapolation, or grammar induction, as the case may be. A fourth column is included in the

----
Insert Table 1 about here
----

table to show how cryptarithmetic can be interpreted as a rule induction task. Information for constructing these processes must either be derived from the experimental instructions or brought into the laboratory from his past experience with the same or analogous tasks (cf. Newell & Simon, 1972, pp. 847-867; Simon & Siklóssy, 1972, Part II).

In concept attainment experiments, for example, the subject is usually specifically instructed as to what concepts are admissible--that is, as to the definition of the space of rules. He is also provided with an explicit definition of the space of possible instances. He must devise a process for testing whether a rule is consistent with the experimenter's classification of an instance, and he must devise a rule generator.

In contrast, in sequence extrapolation tasks much more is usually left to the subject. The space of rules and the rule generator are not usually discussed explicitly in the instructions, nor _a fortiori_, the test for the adequacy or correctness of extrapolation. The experimenter provides the instances and an incomplete statement of the goal (that the sequence is to be extrapolated), and little else. The subject evolves the rest: the space of rules and the rule generator, the process for extrapolating with the rules, and the test for correctness of an extrapolation.

Figures 8 and 9 show programs implementing the GRI framework, for performing the concept attainment and grammar induction tasks, respectively. (The programs have actually been debugged in SNOBOL, but are presented here in a more readable language.) Corresponding routines have the same names in the two programs to make the comparison easy. The particular concept attainment paradigm illustrated allows the subject to generate successive instances and contains a fairly sophisticated rule generator, in order to illustrate the whole range of possible information flows. For programs fitting other concept attainment paradigms see Gregg & Simon (1967).

Insert Figures 8 and 9 about here

The relation of programs like that of Figure 8 to human behavior in the concept attainment task has been examined in some detail in Bruner, Goodnow & Austin (1956), and in Gregg & Simon (1967). Likewise, we have now gathered some data on human behavior in the grammar induction task which fits the scheme of Figure 9 relatively well, but we will have to postpone our analysis of these data to another paper. Our main aim here is to gain an understanding of the relations among these various task environments in terms of problem spaces and channels for the flow of information.

Conclusion

In this paper, we have proposed a conceptualization of problem solving and of rule induction that allows these two arenas of human thinking to be brought within a common framework. We have seen that both problem domains can be interpreted in terms of problem spaces and information processes for searching such spaces. What chiefly distinguishes rule induction tasks from problem solving tasks is that the former call for a pair of problem spaces--one for rules and one for instances--while the latter commonly require only a single problem space. Our analysis of the cryptarithmetic task shows it to lie midway between the two main classes, and hence to provide a useful bridge for translating each of them in terms of the other.

References

Biermann, A. W. and J. A. Feldman. A survey of results in grammatical inference. Conference on Frontiers in Pattern Recognition. Honolulu: January 1971.

Bruner, J. S., J. J. Goodnow and G. A. Austin. A Study of Thinking. New York: Wiley, 1956.

Ernst, George W. and Allen Newell. GPS: A Case Study in Generality and Problem Solving. New York: Academic Press, 1969.

Feldman, Julian, Frederick Tonge and Herschel Kanter. Empirical explorations of a hypothesis testing model of binary choice behavior. In A. Hoggatt and F. Balderston (eds.), Symposium on Simulation Models. Cincinnati: Southwestern Publishing Company, 1963, pp. 55-100.

Gregg, Lee W. and Herbert A. Simon. Process models and stochastic theories of simple concept formation. J. Math. Psychol. 4: 246-276 (1967).

Hunt, Earl. Concept Learning. New York: Wiley, 1962.

Hunt, Earl. Computer simulation: artificial intelligence studies and their relevance to psychology. Annual Review of Psychology 19: 135-168 (1968). [Palo Alto: Annual Reviews, Incorporated]

Klein, Sheldon and Michael A. Kuppin. An intermediate heuristic program for learning transformational grammars. Computer Science Department, University of Wisconsin, Madison. Technical Report #97, August 1970.

Newell, Allen. Heuristic programming: ill-structured problems. In J. S. Aronofsky (ed.), Progress in Operations Research 3: 363-414 (1968). [New York: Wiley]

Newell, Allen. A. I. and the concept of mind. In R. Schank & K. Colby (eds.) Computer Models of Thought and Language. San Francisco: W. H. Freeman, 1973.

Newell, Allen and Herbert A. Simon. Human Problem Solving. Englewood Cliffs, New Jersey: Prentice-Hall, 1972.

Siklóssy, Laurent. Natural language learning by computer. In H. A. Simon and L. Siklóssy (eds.) Representation and Meaning. Englewood Cliffs, New Jersey: Prentice-Hall, 1972, pp. 288-328.

Simon, Herbert A. Complexity and the representation of patterned sequences of symbols. Psychol. Rev. 79: 368-382 (1972).

Simon, Herbert A. The theory of problem solving, Information Processing 71. Amsterdam: North-Holland, 1972, pp. 261-277.

Simon, Herbert A. and Kenneth Kotovsky. Human acquisition of concepts for serial patterns. Psychol. Rev. 70: 534-546 (1963).

Simon, Herbert A. and Laurent Siklóssy (eds.), Representation and Meaning. Englewood Cliffs, New Jersey: Prentice-Hall, 1972.

Solomonoff, Ray. A new method for discovering the grammars of phrase structure languages. Information Processing 59. Proceedings of the International Conference on Information Processing. Paris: UNESCO, 1959.

Woodworth, Robert S. Experimental Psychology. New York: Henry Holt, 1938.

Acknowledgment

*This work was supported by Public Health Service Grant MH-07722, from the National Institute of Mental Health.

```
  1
5ONAL5          D=5
GERAL5          T=0
ROBERO          R > 5, odd
```

Figure 1

A Knowledge State
in a Cryptarithmetic Task

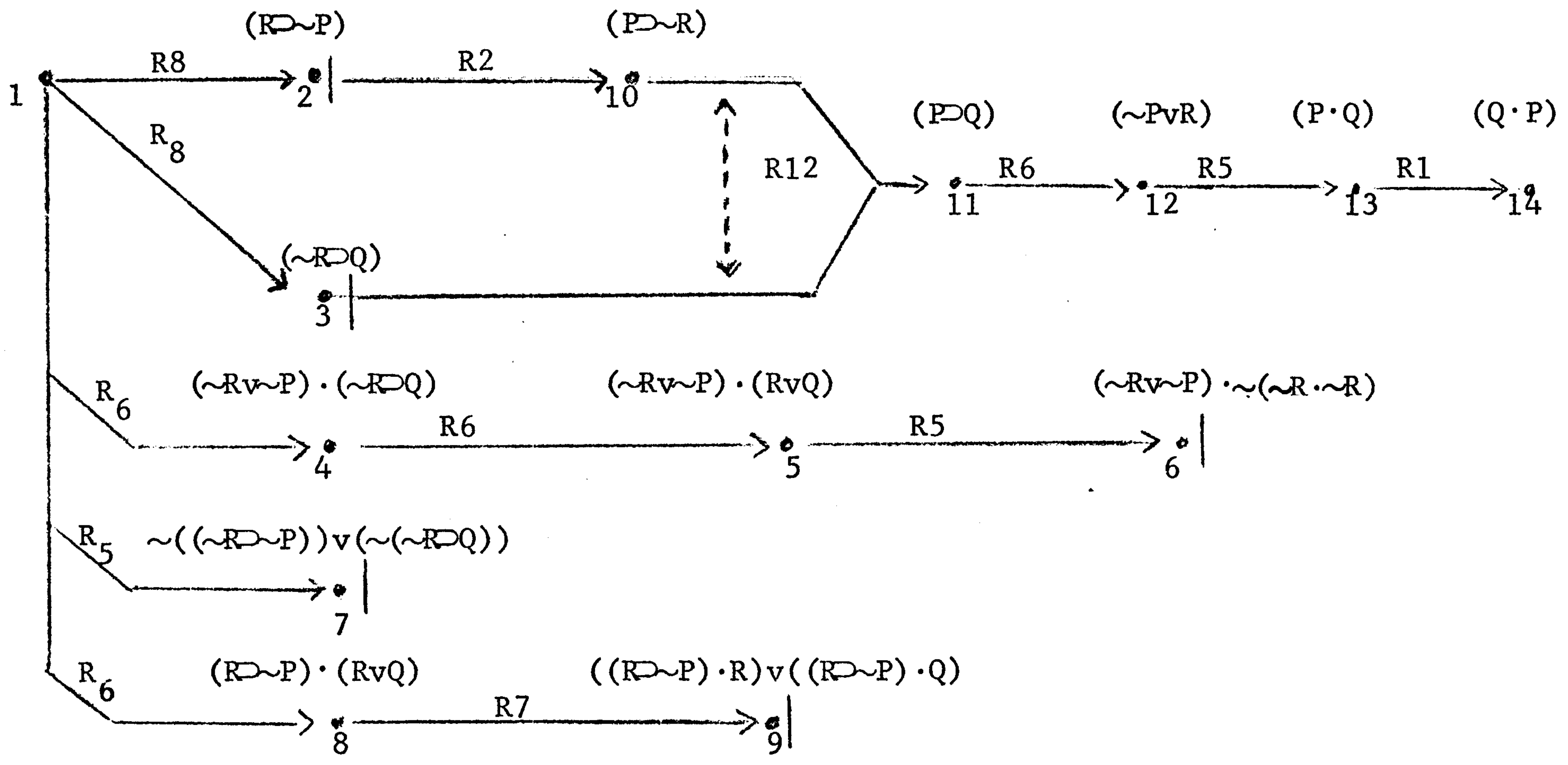

Figure 2

Search Tree Generated by GPS in Logic

Initial expression (node 1) was (R⊃~P)·(~R⊃Q).

Above each node (knowledge state) is shown the new expression that has been derived here.

Below each node is shown the order in which it was generated.

On each link is shown the operator used to generate the next node.

(See Newell & Simon, 1972, pp. 420-425.

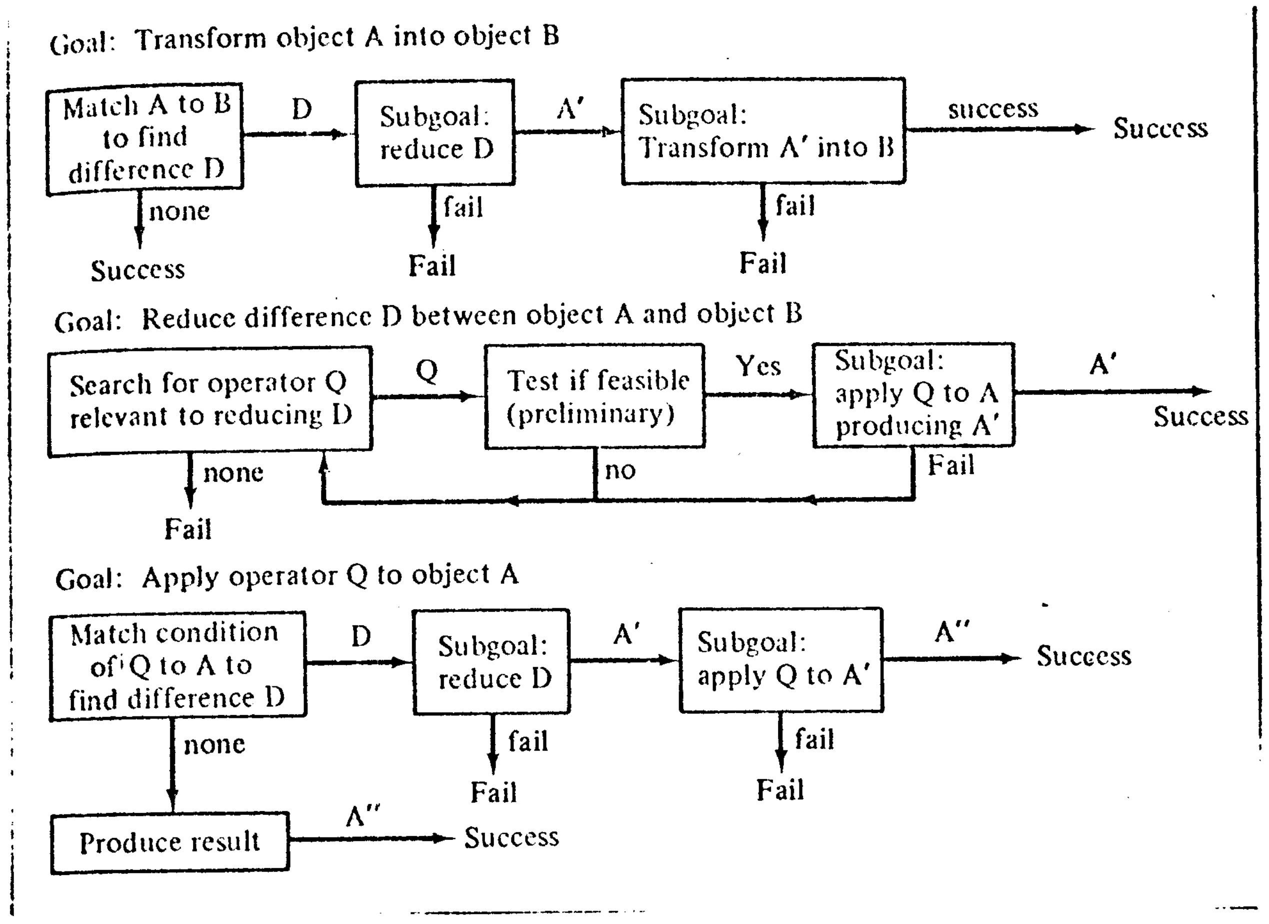
Goal: Transform object A into object B
Match A to B to find difference D
D
Subgoal: reduce D
A'
Subgoal: Transform A' into B
success
Success
none
Success
fail
Fail
fail
Fail
Goal: Reduce difference D between object A and object B
Search for operator Q relevant to reducing D
Q
Test if feasible (preliminary)
Yes
Subgoal: apply Q to A producing A'
A'
Success
none
Fail
no
Fail
Goal: Apply operator Q to object A
Match condition of Q to A to find difference D
D
Subgoal: reduce D
A'
Subgoal: apply Q to A'
A''
Success
none
fail
Fail
fail
Fail
Produce result
A''
Success

Figure 3

GPS Methods - Flow Diagram

(Reprinted with permission from Newell & Simon, 1972, Figure 8.7,p.417)

$$\begin{array}{r} D\ O\ N\ A\ L\ D \\ +\underline{G\ E\ R\ A\ L\ D} \\ R\ O\ B\ E\ R\ T \end{array}$$

Problem Display
(Instance Space)

D=5

List of Assignments
(Rule Space)

Figure 4

Dual Problem Space Interpretation
of Cryptarithmetic Task

$$
\begin{aligned}
2D &= T + 10C2 \\
C2 + 2L &= R + 10C3 \\
C3 + 2A &= E + 10C4 \\
C4 + N + R &= B + 10C5 \\
C5 + O + E &= O + 10C6 \\
C6 + D + G &= R
\end{aligned}
$$

Figure 5

Space of Instances
in Cryptarithmetic

(Showing interdependency of instances
by virtue of carries (C2-C6))

```
General Rule Inducer:
1.  modify rules  (⇒ rules);
    generate instances (⇒ instance):
        classify instance by rules (⇒ instance-class);
        test instance-class (⇒ test-result).
    if tally-correct=criterion exit,
    else go to 1.
```

Figure 6

Sequential Process

for the

General Rule Inducer

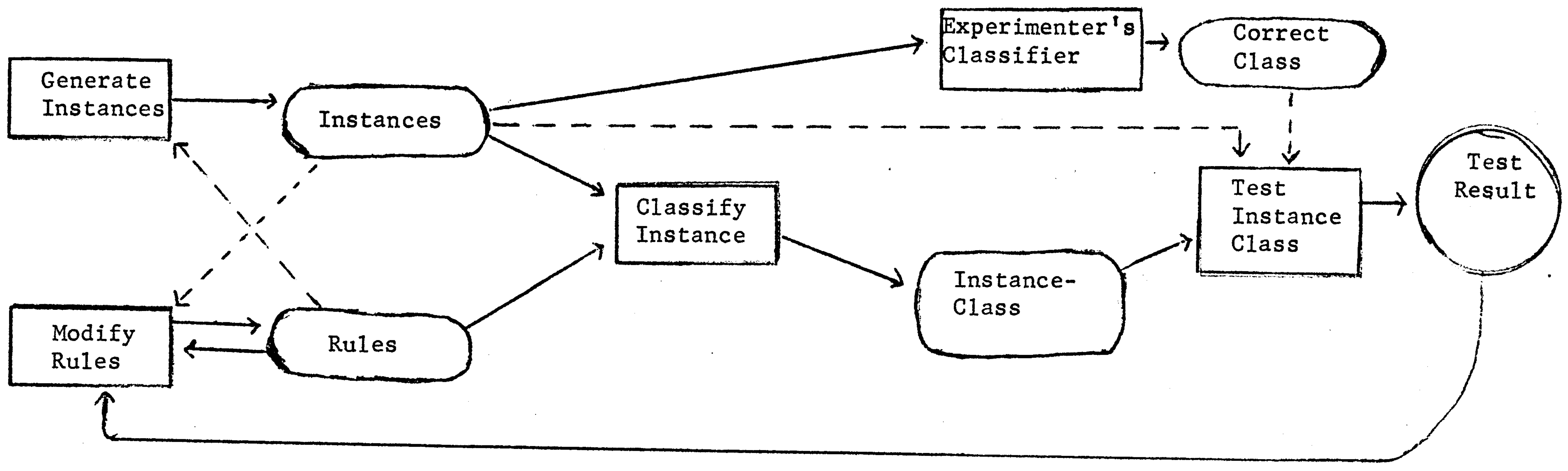

Figure 7

Information Flows

in the

General Rule Inducer

Broken lines show Information Channels used in some, but not all, Variants of the Induction Task.

```
Modify rules (⇒ rules):
   set tally = 0;
1. delete rules from hypothesis-list;
   select item randomly from
      hypothesis-list (⇒ rules);
   classify instance (⇒ instance-class);
   test instance-class (⇒ test-result);
   if test-result = `right´ exit,
      else go to 1.

Generate instances (⇒ instance):
   if parity = `odd´ set parity = `even´,
      else set parity = `odd´;
   produce instance randomly
      from instance-description (⇒ instance);
   classify instance (⇒ instance-class);
   if parity = `even´
      then if instance-class = `positive´ exit,
         else set complement(rules) = rules in instance & exit;,
      else if instance-class = `negative´ exit,
         else set rules = complement(rules) in instance & exit.

Classify instance (⇒ instance-class):
   if rule ∈ instance set instance-class = `positive´ & exit,
      else set instance-class = `negative´ & exit.

Test instance-class (⇒ test-result):
   if correct-rule ∈ instance
      set correct-class = `positive´,
         else set correct-class = `negative´;
   if instance-class = correct-class
      set test-result = `right´
      set tally = tally +1 & exit,
         else set test-result = `wrong´ & exit.
```

Figure 8

Program for Concept Attainment Task

(Subroutines for Executive Program of Figure 6)

```
Modify rules (⇒ rules):
   if test-result = `wrong´
      delete new-rule from rules;
   if basic-sentence-tally = `done´ go to 1,
      else generate basic-sentence (⇒ rule)
         set new-rule = rule & exit;
1. if replacement-rule-tally = `done´ exit,
      else generate replacement-rule (⇒ rule) &
         set new-rule = rule & exit.

Generate instances (⇒ instance):
   if new-rule ∈ basic-sentences
      set instance = new-rule & exit,
      else generate item from positive-instance (⇒ item);
         apply new-rule to item (⇒ instance)
         if instance ∉ positive-instance exit,
            else continue generation;
         if positive-instances exhausted
            set signal = `finished´ & exit.

Classify instance (⇒ instance-class):
   generate basic-centences (⇒ basic-sentence):
      if instance = basic-sentence set instance-class = `positive´
         & exit from routine,
         else continue generation;
   generate derived-sentences with length = length(instance)
         (⇒ derived-sentence):
      if instance = basic-sentence set instance-class = `positive´
         & exit from routine,
         else continue generation;
   set instance-class = `negative´ & exit.
```

Figure 9 (continued →)

```
Test instance-class (⇒ test-result):
    if instance ∈ legitimate-instances add instance
        to positive-instances & set correct-class = `positive´,
        else set correct-class = `negative´;
    if instance-class = correct-class
        set test-result = `right´ &
        set tally = tally + 1 & exit,
        else set test-result = `wrong´ &
            set tally = 0 & exit;
    add instance to tested-instances & exit.
```

Figure 9

Program for Grammar Induction Task

(Subroutines for Executive Program of Figure 6)