Chapt. 5.5

Published in L. W. Gregg (ed.)
KNOWLEDGE AND COGNITION.
Potomac, Maryland: Lawrence
Erlbaum Associates, 1974.

5

PROBLEM SOLVING AND RULE INDUCTION: A UNIFIED VIEW

(with
~~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 "concept attainment," "pattern induction," or "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:

> 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 [Woodworth, 1938, p. 746].

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 artificial 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

This work was supported by Public Health Service Grant MH-07722, from the National Institute of Mental Health. We are grateful to James Greeno and Allen Newell for helpful comments on an earlier draft of this paper.

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. To be sure, 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 *discover* this deduction, this proof; and the discovery process, which is the problem-solving process, is wholly inductive in nature. It is a search through a large space of logic expressions for the goal expression—the theorem. 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, 1972c) give us a clue as to how to go about building a common body of theory that will embrace both problem solving and rule induction, including concept attainment. It is the aim of this paper to outline such a theory. We shall not adduce new empirical evidence, nor even refer to particular experiments in the literature. Instead, we shall take as our starting points the recent formulation of the theory of problem solving mentioned above (Newell & Simon, 1972), and a recently formulated and rather general process model of concept attainment (Gregg & Simon, 1967), and show how both of these relate to the more general framework that is our goal. Since these theories have substantial empirical underpinnings, the discussion will be tied firmly to empirical data, albeit indirectly.

PRELIMINARY REMARKS

Before proceeding, we need to say more clearly what we mean by "common body of theory." A theoretical explanation of the behavior of a subject confronted with a problem-solving task or a concept-attainment task might take the form of a program, an organization of information processes, more or less appropriate to carrying out the task. This is, in fact, the form of the problem-solving theory of Newell and Simon and the concept attainment theory of Gregg and Simon mentioned in the last paragraph. To the extent that two programs explaining behavior in these two kinds of task environments employ the same basic processes, or to the extent that the processes are organized isomorphically, we may say they express a common theory.

But we must be more specific about what is common to them. The fact that two physical theories can both be stated in terms of differential equations connects them only superficially. Even less should we be surprised or impressed to find that two theories of human information processing performance can be written in the same programming language. Computer languages—IPL-V, LISP, SNOBOL—are almost completely general, capable of describing any organization of information processes. Anything that can be done by a Turing Machine can be described in any of these languages. When we speak of a common theory for problem solving and rule induction we intend to assert more than that man is a Turing Machine.

Nor is it sufficient—or very informative—to show that it is possible to write a single program that will simulate and describe human behavior in both a problem-solving and a concept-attaining environment. That kind of generality could be achieved by a "big switch"—a pair of subprograms joined only by a simple test to identify the task environment, and to select from the pair the appropriate subprogram to deal with it.

The generality we seek, then, is not the nearly vacuous generality of either the Turing Machine or the Big Switch. Our aim is to show a much closer relation between problem-solving processes and rule-inducing processes than is implied by either of these. Exactly what this means will become clear as we proceed.

Because "problem solving" and "rule induction" are themselves heterogeneous domains with ill-marked boundaries, we will make matters more concrete by referring to some specific illustrative tasks. For problem solving, we will pay special attention to two tasks analyzed at length in Newell and 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; Gregg & Simon, 1967; Hunt, 1962), extrapolation of serial patterns (Feldman, Tonge, & Kanter, 1963; Simon & Kotovsky, 1963; Simon, 1972a), and induction of the rules of a grammar (Klein & Kuppin, 1970; Solomonoff, 1959; Siklossy, 1972).

Our undertaking is a little more ambitious than has been indicated thus far. 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, previous theoretical treatments of concept attainment do not include extrapolation of patterned sequences, and theories of sequence extrapolation do not encompass the standard experimental paradigms for studying concept attainment. Here we will aim at a unified treatment of the whole range of things we have here been calling "rule induction," and a comparison of these, in turn, with the activities called "problem solving."

We will begin by outlining the basic features of the information processing theory of problem solving, and then use these features to construct the broader theory.

PROBLEM SOLVING

In solving a well-structured problem (and this is the only kind we shall deal with), the problem solver operates within a *problem space.* A problem space is a set of points, or nodes, each of which represents a *knowledge state.* A knowledge state is the set of things the problem solver knows or postulates when he is at a particular stage in his search for a solution. For example, at a certain point in his attempt to solve the cryptarithmetic problem, DONALD +GERALD=ROBERT, the problem solver may know that the number 5 must be assigned to the letter D, the number 0 to T, the number 9 to E; and he may know also that R is odd and greater than 5. The conjunction of these bits of knowledge defines the particular node he is currently at in his problem space, and the space is made up of a collection of such nodes, each representing some set of pieces of knowledge of this kind.

Problem-solving activity can be described as a search through the space (or maze, or network) of knowledge states, until a state is reached that provides the solution to the problem. In general, each node reached contains a little more knowledge than those reached previously, and the links connecting the nodes are search and inference processes that add new knowledge to the previous store.

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 encoded addition problem. The problem solver moves from one state to another by inferences (or conjectures) 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

FIG. 5.1. A knowledge state in a cryptarithmetic task. (The figure shows what the problem solver knows after his initial processing of the sixth, fifth, and first columns of the display.)

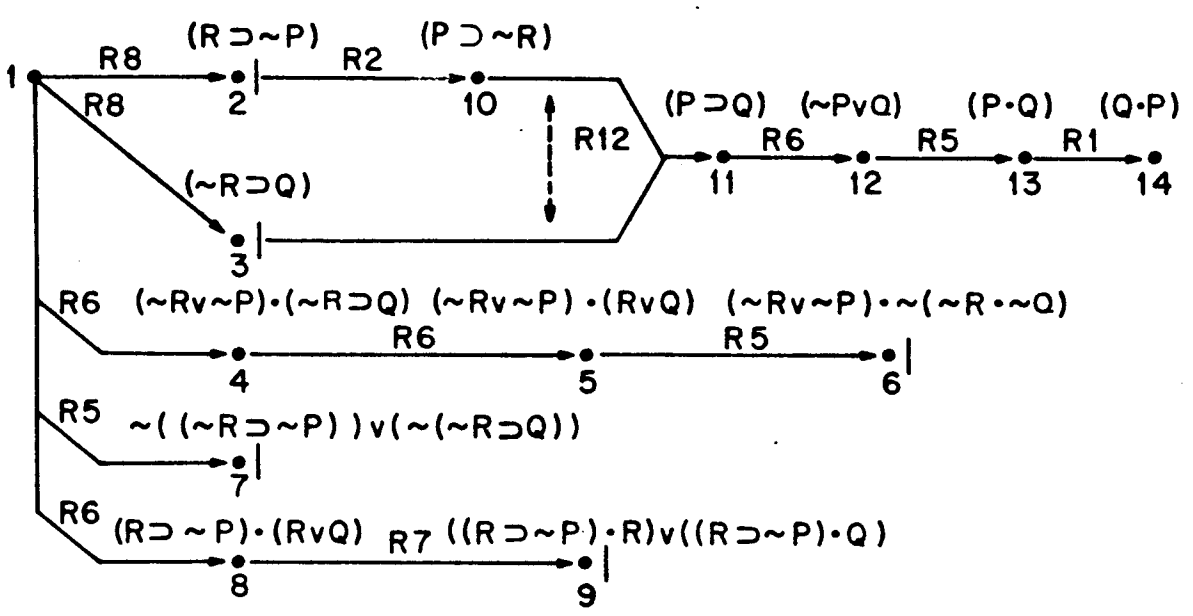

FIG. 5.2. Search tree generated by GPS in logic. (Initial expression (node 1) was (R ⊃~ P) · (~R ⊃ Q). Above each node (knowledge state) is shown the new expression that has been derived here. Below each node is shown the order in which it was generated. On each link is shown the operator used to generate the next node. See Newell & Simon, 1972, pp. 420-425.)

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

The search through such a problem space is generally highly selective, being 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: (*a*) selection, from among those already visited, of a particular knowledge state from which to continue his search and (*b*) selection of a particular operator (inference rule, or "move") to apply at that node in order to reach a new knowledge state.

Means-ends analysis, which appears to be 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 analysis, the information in a particular knowledge state that has already been reached 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.

We may formalize and generalize this description of problem solving as follows:

1. There is a *problem space* whose elements are *knowledge states.*

2. There are one or more *generative processes* (operators) that take a knowledge state as input and produce a new knowledge state as output.

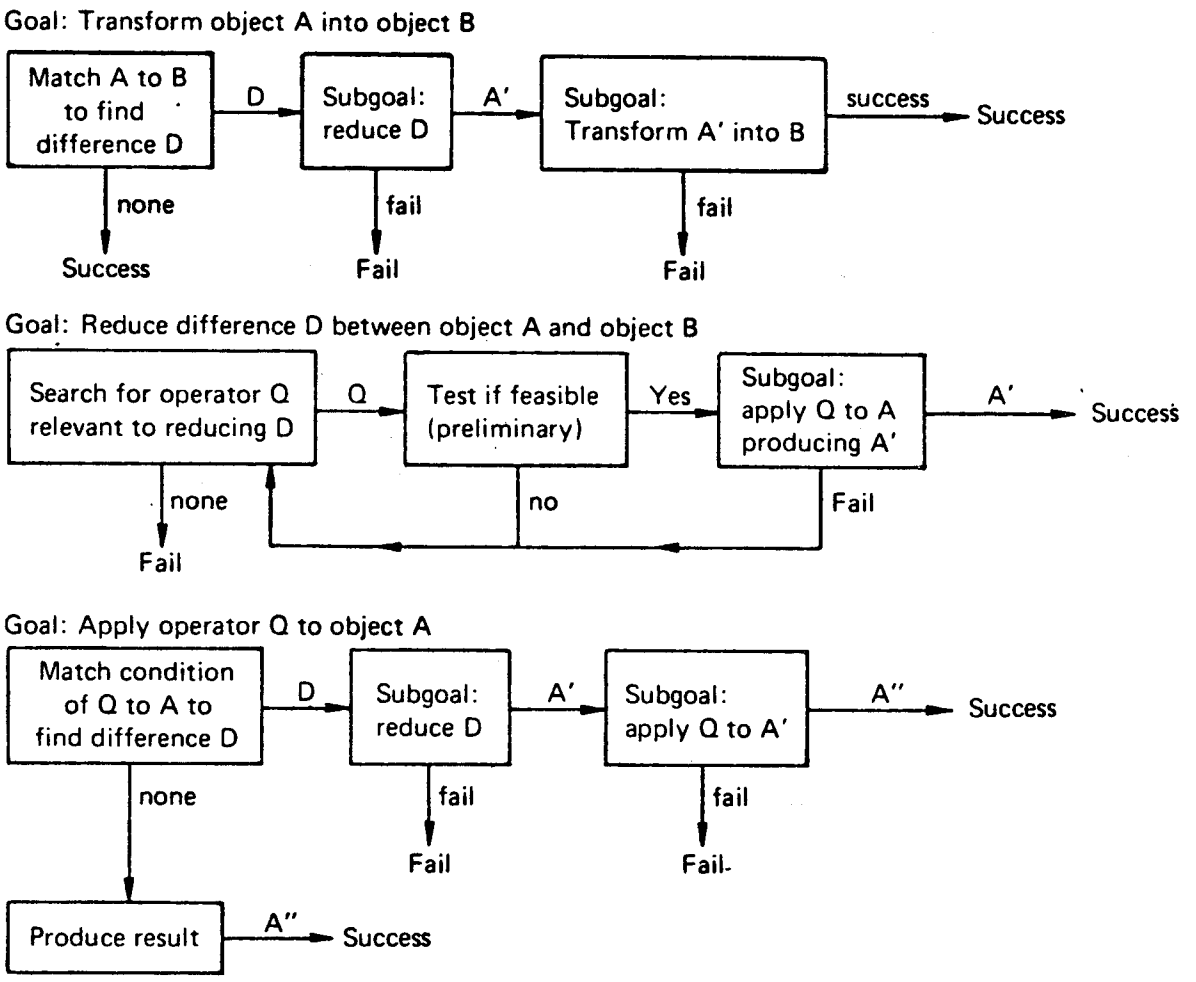

FIG. 5.3. GPS methods—flow diagram. Information in the form of differences between the current knowledge state and the goal is used to select operators that may reduce the differences. (Reprinted with permission from Newell & Simon, 1972, Fig. 8.7, p. 417.)

3. 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.

4. There are processes for *selecting* which of these generators and tests to employ, on the basis of the information contained in the knowledge states.

The crucial points in this characterization are the third and fourth postulates: 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. The problem-solving process is an information gathering process as much as it is a search process. The accumulation of information in the course of search permits the search to be selective, and gives problem solving in very large problem spaces a chance of success. The processes for using this information to steer the search are generally processes of inductive inference. Being inductive, they do not provide certainty, but have only heuristic value in guiding the search and making it efficient.

Characterizing problem solving as information gathering gives us the framework we need to deal with the whole range of tasks in which we are

interested. We shall describe the process for all of these tasks as a search through a problem space guided by information accumulated during the search. And we shall undertake to show that the fundamental search processes (generation, test, and selection processes), as well as the inference processes, are of the same kind in rule induction tasks as in problem-solving tasks, and are organized in a very similar way. Finally, we shall see that the basic *difference* between the two domains is that rule induction involves an alternation of activity between two distinct, but interrelated, problem spaces, while only a single space is involved in problem solving.

Information Gathering in Theorem Proving

Consider the following GPS-like system for discovering proofs for theorems in symbolic logic. Many subjects in the laboratory have been observed to follow essentially this process. The knowledge states are sets of logic expressions that have been derived from the initial premises. Two kinds of information are used to guide the search: (*a*) the degree of similarity or difference between the expressions contained in a given knowledge state and the goal expression and (*b*) 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 (see Fig. 5.3).

Information Gathering in Cryptarithmetic

We shall use the cryptarithmetic task as a "bridge" from the topic of problem solving to the topic of rule induction because it is possible to give an interpretation to the task which places it in either of the two categories. Although 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, we shall look at matters in a slightly different way. Let us consider the knowledge states in cryptarithmetic to be made up of two distinguishable components: the *problem display* in which digits have replaced those letters to which assignments have already been made and the *list of assignments* themselves. The problem-solving goal can then be described in two ways: (*a*) to replace all letters in the display by digits in such a way that the resulting problem in arithmetic is correct or (*b*) 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 to

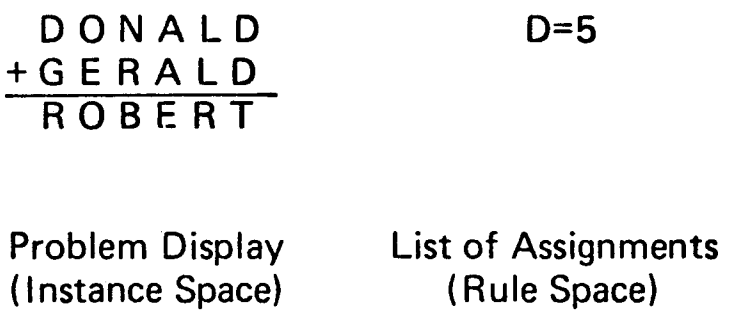

FIG. 5.4. Dual problem space interpretation of cryptarithmetic task.

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, then 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 by applying simple arithmetic processes. 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" 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 assigned digit for the corresponding letter whenever the latter occurs in the display. For example, suppose we have the display 5ONAL5+G9RAL5= ROB9R0 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 = 7 v 9. Then, if there is a process for examining the consistency of assignments, that process can draw the inference that R = 7, and replace R = 7 v 9 on the list by this more precise assignment. Similarly, 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 sets of assignment *rules* (rules for substituting digits for letters in the display), and a space of sets of *instances* (columns of the display). The goal is to complete the set 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 (Fig. 5.5). 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 the cryptarithmetic task from the category of problem solving to the category of concept attainment, pattern induction, or rule discovery.

From this example, we hypothesize 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 at each of the nodes 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 obtaining and utilizing information, we can provide the common framework that we have been seeking for all of these tasks.

RULE INDUCTION

If the theory of rule induction is to bear a close relation to the theory of problem-solving processes, then it must be constructed of the same basic modules: one or more generating processes, one or more test processes, and one or more processes to select the generators and tests to be applied, and to determine the order of their application. Newell (1965, 1973) has proposed a

```
          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
```

FIG. 5.5. Space of instances in cryptarithmetic. (Showing interdependence of instances by virtue of carries, C2-C6.)

taxonomy of general problem-solving methods that lists the principal ways in which these modules can be combined into operative systems. By "general methods" Newell means methods that make relatively unspecific demands upon the task environment, and hence, are widely applicable.

Some General Methods

We will be concerned with just three of the methods Newell defines: the generate-and-test method, the heuristic search method, and the induction (or hypothesis-and-match) method. We shall see that the first two of these are characteristic of problem-solving systems, the third of rule induction systems, but that they differ mainly with respect to the information flows among the modules. All of these methods may draw upon one or both of two submethods: the matching method, and the means-ends method.

At a minimum, any goal directed system must include a generator for producing new knowledge states and a test for determining whether a state produced by the generator is in fact a solution state. The simplest solution method is just this minimal *generate-and-test.* The power and efficiency of the method derives from information that is implicit in its structure. If, for example, the generator can produce only a very small set of states, and if this set is guaranteed to contain a solution, the method will be powerful, for the solution will be found promptly. If the test can reject inadequate solutions rapidly—say, by means of a matching process—then the cost of testing will be relatively small.

In the generate-and-test method, the order in which nodes are generated is independent of the knowledge that is gradually accumulating—the information is used only by the test process. Consider next a more sophisticated system, where the generator is no longer insensitive to knowledge that has been produced. Now information flows back from the test to the generator. This feedback requires the test to provide more information than just the success or failure of the match between the knowledge states generated, and the specification of the desired knowledge state (the goal). Using the test information, the generator produces a new knowledge state by modifying a state produced previously in the search. This dependency of generation upon the test outcome characterizes the *heuristic search* method.

We have already remarked on two kinds of information that can be used by the generator in heuristic search: first, information to select which of the previously generated states will be modified to produce the next state; second, information to select which of several available operators will be applied to the knowledge state to modify it. If the latter choice depends on the test's detecting specific differences between a state and the goal state, then we speak of the *means-ends* submethod.

Thus far everything that has been said applies equally well to problem solving and to rule induction. In the former case, the search ends with the discovery of the problem solution; in the latter, with the discovery of a rule that is consistent with a set of instances. In both cases, the key process is a search—an inductive process. The search for a rule can be (and usually will be) a heuristic search, and can employ the means-ends submethod, as we shall see.

What distinguishes rule induction tasks from problem-solving tasks is the nature of the test process. In a rule induction task, the attainment of a solution is determined by applying the proposed rule to objects (*instances*), and by then testing whether the application gives a correct result. The test is not applied directly to the rule, but to another set of expressions, the instances. The evaluation of the rule thus takes an indirect path, and the feedback of information from test to generator retraces this path. A rule is rejected or modified if false instances are associated with it, or if there exist true instances that are not associated with it.

In a rule induction task we can define a space of sets of instances in addition to the space of sets of rules. The test process for the rule induction system operates within the space of instances. It can incorporate an instance generator (unless the instances are generated by the experimenter), as well as instance tests (which may or may not make use of knowledge of results provided by the experimenter). Suppose that the overall test process contains both generator and test subprocesses operating in the space of the instances. These subprocesses and their organization may, in turn, exhibit various levels of sophistication in their use of information—e.g., in the feedback of information from the test subprocess to the generator subprocess. A primitive test process would employ only the generate-and-test method; a more powerful one, heuristic search, possibly including the means-ends submethod.

In Newell's taxonomy, a system uses the induction method if there are separate generators for rules and instances, and a match process to test whether an instance agrees with (is associated with) a rule. Since all heuristic search methods are inductive, as we have seen, it will be better to refer to this method as the *rule induction* method. It is clear that the rule induction method, so defined, is really a whole collection of methods. Nor is the locus of variation limited to the test process, as sketched in the last paragraph. There can also be various arrangements for the flow of information *between* the instance space and the space of rules—i.e., between the test process and the generator process of the entire rule induction process (see Fig. 5.6).

In the most primitive system, there is no feedback of information from test to rule generator (Channel *e*, Fig. 5.6); 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 a rule induction version of the generate-and-test method, adapted to the dual problem space.

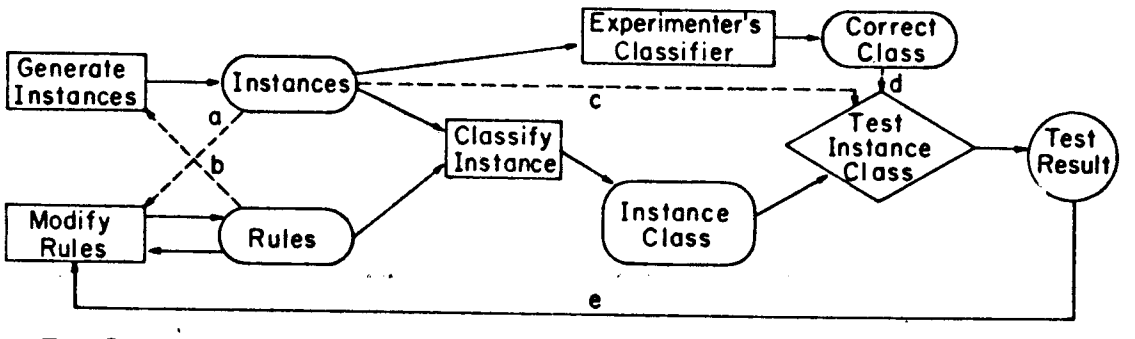

FIG. 5.6. Information flows in rule induction processes. (Broken lines show information channels used in some, but not all, variants of the rule induction task.)

On the other hand, if the rule generator does not create each rule anew, but produces it by modifying previous rule sets on the basis of information received from the test of instances (Channels *a* and *e*. Fig. 5.6), then we have a rule induction version of the heuristic search method.

Further, the existence of two 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 need not be autonomous, but may instead derive information from the rules that have been generated and the previous tests that have been performed—a flow of information from rule space to instance generator (Channel *b*, Fig. 5.6) as well as from instance space to rule generator (Channel *a*). 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 just what is happening in the cryptarithmetic solution method described earlier when we view the columns of the problem display as instances, and the list of assignments as a list of rules.

A General Rule Induction Program

We are now ready to define a formal system that expresses the common theory we are seeking. Fig. 5.7 gives the definition of a General Rule Induction (GRI) Executive Program. In order to make it as readable as possible, the definition is expressed in the informal programming language defined in Newell and Simon (1972, pp. 38-51).

The GRI system is extremely simple, consisting of a subprocess to generate rules, and a second subprocess to generate and test instances. The output of the test (*test-result*) is available as an input to the rule generator, to help guide the next step of generation. Whether this information will be used, and in what way, depends on the internal structure of the rule generator, which is not specified. Thus GRI employs the generate-and-test method. Whether it employs heuristic search or even more elaborate methods depends

on the specification of the sub-processes and the information flows between them. Notice that just as the test results are available as input information to the rule generator, so the set of rules is available as input information to the instance generator.

Fig. 5.7 makes patent that the only feature distinguishing a rule induction system from a problem-solving system is that the tests of the rule induction system operate in a different space from the generator. Generator and test use the same space in a problem-solving system. The fundamental generator-test alternation is identical for both kinds of systems, but the similarity between them extends much further. In both problem-solving and rule-induction systems, the selectivity of generators depends upon the feedback of information from the test processes. Because the rule induction system may contain two generators, rather than just one, there is a larger number of possible channels of information flow, hence, a richer taxonomy of possible specialized systems.

In Fig. 5.6, we have shown the flows of information in GRI. Some of these (shown by broken lines) are "optional," in the sense that variants can be devised that include or exclude them. We will illustrate this point in the next sections, when we discuss how GRI would handle some of the standard paradigms for concept attainment, series extrapolation, and grammar induction.

GRI is capable of performing the whole range of tasks just mentioned. We must be careful as to what we mean by this claim. 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 the same as the situation confronting the General Problem Solver. GPS is a general organization for performing means-ends

```
General Rule Inducer:
1. generate rules (⇒ rules);
   generate instances (⇒ instance);
      classify instance by rules (⇒ instance-class);
      test instance-class (⇒ test-result).
      if test-result = 'correct' tally = tally + 1,
         else set tally = 0;
   if tally = criterion exit,
      else go to 1.
```

FIG. 5.7. Executive program for the General Rule Inducer.

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. The General Rule Inducer needs the same kinds of problem specification in order to tackle specific tasks. GRI itself is an executive program providing an organization within which the specialized subprocesses can operate. We now present some examples of such specialized subprocesses that are applicable to the specific task domains of concept formation, series extrapolation and grammar induction.

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"). 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 searches 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.

The behavior of subjects in concept attainment tasks of the kinds studied by Bower and Trabasso and others has been formalized by Gregg and Simon (1967) in a family of programs whose individual members differ only with respect to the amount of information the subject is assumed to retain as a basis for guiding the concept generator (Channels *a* and *e* of Fig. 5.6). The programs described by Gregg and Simon conform to the organization of the GRI executive.

In those variants of the program where no information is fed back (Channel *a* inoperative), whenever a guess has been wrong, the generator selects a concept at random from the set of available concepts. A slightly more efficient generator (which, strictly speaking, requires feedback via Channel *e* only to signal whether or not the last instance was classified correctly) samples randomly from the set of available concepts, but without replacing those already eliminated. A somewhat more efficient generator, using also feedback via Channel *a*, produces a concept consistent with the correct classification of the most recent instance. A still more efficient generator produces a concept consistent with the classifications of all previous instances. Empirical data in the literature indicate, according to Gregg and Simon, which of these methods will be employed by a human

subject depends 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 described by Gregg and Simon's program does not incorporate a flow of information from the space of concepts to the generator of instances (Channel *b* in Fig. 5.6), since the instances in those experiments 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 (*Classify Instance,* Fig. 5.6 and 5.7) 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, in other forms of the concept attainment experiment (Bruner et al., 1956) 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 (Channel *b*), enables solution methods that are more efficient than any 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.

The programs of Gregg and Simon do not cover the concept attainment paradigm in which the subject selects the instances. However, it is easy to generalize their programs to cover this case within the executive program of Fig. 5.7. A set of processes that accomplishes this is shown in Fig. 5.8. Each of the four processes—modify rules, generate instances, classify instance, and test instance-class—is extremely simple. The rule generator and instance generator embody particular assumptions about the subject's strategy for using information to enhance selectivity. The rule generator remembers which hypotheses have already been rejected, and also requires the new hypothesis to be consistent with the previous instance. The particular instance generator that is provided here generates instances that are positive for the current rule on half the trials and negative on the other half. As a guarantee of the completeness of the analysis, a SNOBOL version of the program of Fig. 5.7 and 5.8 has been written and tested. By modifying the several processes in simple ways, always employing the executive of Fig. 5.7, a wide range of experimental paradigms and of subject strategies within each of those paradigms can be simulated.

```
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.
```

FIG. 5.8. Program for concept attainment task. (Subroutines for executive program of Fig. 5.7.)

Extrapolation of Patterned Sequences

A theory of how human subjects discover the patterns implicit in sequences of letters or numbers and use these patterns to extrapolate the sequences was developed in the form of a computer program by Simon and Kotovsky (1963). The relation of this theory to other theories of performance

in this task and to the empirical data has been reviewed by Simon (1972a). The pattern discovery program is also an instance of the schema of Fig. 5.7.

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 fill in the blanks "appropriately," we must employ the notions 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 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 letter is *next* (N) in the English alphabet to the second letter (2) in the previous period (p); the second letter in each period is next (N) to the first letter (l) in the same period (s); the third letter in each period is the constant letter 'M', i.e., is the *same* (S) as the third letter (3) in the previous period (p). The pattern might be described as 'N2p Nls S3p'. The sequence is initialized by supplying the beginning 'A' and the constant 'M'.

Clearly, the elements of the sequence itself in the extrapolation task are the counterparts of the instances in the concept attainment task; while the pattern is the counterpart of the concept. What are the flows of information? As in the simple 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 (Channel *b*, Fig. 5.6) 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 (Channel *a*), 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

---

[1]Ernst and Newell (1969) have proposed an ingenious scheme for handling the sequence extrapolation task as a GPS 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. However, their proposal shows again the close affinity between problem solving and rule induction, as these terms are commonly used.

symbols in the sequence, and then hypothesize patterns constructed from those relations (an example of the *matching* method). 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 nonsentences. This task has received some attention in the artificial intelligence literature (e.g., Biermann & Feldman, 1971; Klein & Kuppin, 1970; Siklossy, 1972; Solomonoff, 1959). 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 will be classified as 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 nonsentences 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 (Channel *b*, Fig. 5.6), as well as a reverse flow from instances to rules (Channel *a*).

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 the set of instances of valid sentences in

advance. Or the experimenter could supply instances of sentences and nonsentences, 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.

Fig. 5.9 shows processes for performing the grammar induction task that again operates with the GRI executive of Fig. 5.7. These processes are a little more complicated than those of Fig. 5.8, mainly because they must generate and test two different kinds of rules: basic sentences and replacement rules. The specific generators for the two kinds of rules are not defined in the figure. This program, like the one for concept attainment, has also been written and debugged in SNOBOL (with specific versions of the generators for basic sentences and replacement rules). We have begun to gather some data on human behavior in the grammar induction task which, on first examination, fit the program of Fig. 5.7 and 5.9 relatively well, but we will have to postpone detailed analysis of these data to another paper.

The Tower of Hanoi: A Digression

We digress for a moment to comment on the Tower of Hanoi problem, discussed by Greeno and Egan in their paper for this volume, for this task illustrates again how tricky is the distinction between problem-solving tasks and rule induction tasks. The problem as usually stated—to find a sequence of moves that will transfer all the disks from one peg to another, subject to the usual constraints on moves—is clearly a problem-solving task. If demonstration of this is needed, it has been provided by Ernst and Newell (1969), who programmed GPS to solve the problem by the means-ends method.

But the problem can be stated differently: to find a *rule* for transferring the disks from one peg to another. It may also be required that the rule work properly for an arbitrary number of disks. Just as clearly, this is a rule induction task. To solve it, one or more rule spaces must be formulated and searches conducted through these spaces. Knowledge to guide this search may be obtained by manipulating the disks—that is, by searching through the space of arrangements of disks on pegs. Thus *this* Tower of Hanoi problem, as distinguished from the one described in the previous paragraph, involves a dual problem space.

Rules for the Tower of Hanoi can be stated in various forms. One (incomplete) rule is based on the sequence: 1 2 1 3 1 2 1 4 1 . . . , where the digits refer to the disks to be moved. With slight modification, the sequential pattern programs discussed earlier could discover this pattern. The recursive solution to the problem requires a different kind of rule generator—one that understands the concept of recursive definition.

```
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-sentences (⇒ 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.

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',
      else set test-result = 'wrong';
   add instance to tested-instances & exit.
```

FIG. 5.9. Program for grammar induction task. (Subroutines for executive program of Fig. 5.7.)

Summary: Application of GRI to Specific Task Environments

Table 5.1 shows how we have interpreted the processes of GRI in the context of the specific tasks we have discussed: concept attainment, sequence extrapolation, and grammar induction. A fourth column in the table shows how the cryptarithmetic task can be handled within the same schema when viewed as a rule induction task; while the fifth column shows which components of the schema have counterparts in a problem-solving task where only a single problem space is involved.

We have shown how the specific processes that describe subject behavior within the executive program of GRI vary as a function of the characteristics of the experimental paradigm and the level of complexity and sophistication of the strategy that the subject adopts for handling the task.

In concept attainment experiments, for example, the subject is usually instructed specifically as to what concepts are admissible, that is, he is given the space of rules. He is also provided with an explicit definition of the space of possible instances. 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 the extrapolation. The experimenter provides the instances (the incomplete sequence) and an ill-defined goal (that the sequence is to be extrapolated). The subject evolves the rest: the space of rules and the test for correctness of an extrapolation, as well as the generator and test processes that define his strategy.

Variations in subject strategies relate particularly to the use of information from each of the problem spaces—the rule space and instance space—to guide the generator for searching the other. In paradigms where the experimenter provides one of the generators (e.g., the instance generator in the standard concept attainment paradigm) there is less room for variation in subject strategy than in paradigms where the subject must devise both generators (e.g., in the form of the concept attainment experiment used by Bruner et al., 1956).

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. The generators of elements in a problem space may be more or less selective, depending on what use they make of information provided by the tests, and varying levels of selectivity can be observed in both rule-induction systems and problem-solving systems. What chiefly distinguishes rule-induction tasks from problem-solving tasks is that

TABLE 5.1

Application of GRI to Four Tasks

	GRI	Task environments				
		Concept formation	Sequence extrapolation	Grammar induction	Crypt-arithmetic	Problem solving
Rule space	Rules	Current hypothesis	Partial pattern	Partial grammar	List of assignments	Node in Problem space
	Modify rules	Generate hypothesis	Modify pattern	Modify grammar	Modify list	Apply operator at node
Instance space	Instances	Instances	Sequence elements	Predicted Sentences	Column of display	—
	Generate instance	(Generate instance)	[Generate sequence]	Generate Sentence	Update display	—
Test	Classify instance	Respond	Predict symbol	Query experimenter	Process column	Describe new node
	Test instance-class	[Reinforce response]	Match symbols	[Accept sentence]	Detect contradiction	Evaluate new node

Note. Processes in square brackets are executed by experimenter; processes in parentheses are sometimes executed by experimenter.

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.

To test the conceptualization, and to guarantee that it is more than a set of analogies, we constructed a formalization, the General Rule Induction program, together with subprocesses for concept attainment and grammar induction that operate within that program.[2] By means of GRI, each of the tasks can be mapped formally on the others. The basic components of these programs are generator and test processes organized into generate-test, heuristic search, means-ends, and matching methods.

[2]Since this was written, Dennis E. Egan and James G. Greeno have written SNOBOL routines that operate within GRI for discovering sequential patterns (personal communication). Thus the GRI scheme has now been implemented for the three main rule induction tasks discussed in this paper.

CHAPTER 5.5

REFERENCES

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

Bruner, J. S., Goodnow, J. J., & Austin, G. A. A study of thinking. New York: Wiley, 1956.

Feldman, J., Tonge, F., & Kanter, H. Empirical explorations of a hypothesis testing model of binary choice behavior. In A. Hoggatt & F. Balderston (Eds.), Symposium on simulation models. Cincinnati: Southwestern Publishing Co., 1963.

Gregg, L. W., & Simon, H. A. Process models and stochastic theories of simple concept formation. Journal of Mathematical Psychology, 1967, 4, 246-276.

Hunt, E. B. Concept learning: an informational processing problem. New York: Wiley, 1962.

Hunt, E. Computer simulation: Artificial intelligence studies and their relevance to psychology. Annual Review of Psychology, 1968, 19, 135-168.

Klein, S., & Kuppin, M. A. An intermediate heuristic program for learning transformational grammers. (Tech. Rep. No. 97), Computer Science Department, University of Wisconsin, Madison. August, 1970.

Newell, A. A.I. and the concept of mind. In R. Schank & K. Colby (Eds.), Computer models of thought and language. San Francisco: Freeman, 1973.(a)

Newell, A. Production systems: Models of control structures. In W. G. Chase (Ed.), Visual information processing. New York: Academic Press, 1973.(b)

Newell, A., & Simon, H. A. Human problem solving. Englewood Cliffs, N.J.: Prentic-Hall, 1972.

Siklossy, L. Natural language learning by computer. In H. A. Simon & L. Siklossy (Eds.), Representation and meaning. Englewood Cliffs, N.J.: Prentice-Hall, 1972.

Simon, H. A. Complexity and the representation of patterned sequences of symbols. Psychological Review, 1972, 79, 368-382.(a)

Simon, H.A. The theory of problem solving. Information Processing 71. Amsterdam: North Holland, 1972(c)

Simon, H. A. & Kotovsky, K. Human acquisition of concepts for sequential patterns. Psychological Review, 1963, 70, 534-546.

Solomonoff, R. 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, R. S. Experimental Psychology. New York: Holt, 1938.