DOCUMENT RESUME

ED 201 337 IR 009 351

AUTHOR	Hayes-Roth, Frederick; And Others
TITLE	Knowledge Acquisition, Knowledge Programming, and Knowledge Refinement.
INSTITUTION	Rand Corp., Santa Monica, Calif.
SPONS AGENCY	National Science Foundation, Washington, D.C.
REPORT NO	ISBN-0-8330-0195-7; Rand-R-2540-NSF
PUB DATE	May 80
GRANT	MCS77-03273
NOTE	39p.
AVAILABLE FROM	Rand Corporation, Main St., Santa Monica, CA 90406 ($3.00).
EDRS PRICE	MF01 Plus Postage. PC Not Available from EDRS.
DESCRIPTORS	*Artificial Intelligence; *Computers; Information Theory; Man Machine Systems; *Programing; Research Reports; Systems Development

ABSTRACT

This report describes the principal findings and recommendations of a 2-year Rand research project on machine-aided knowledge acquisition and discusses the transfer of expertise from humans to machines, as well as the functions of planning, debugging, knowledge refinement, and autonomous machine learning. The relative advantages of humans and machines in the building of intelligent systems are explained. Background and guidance is provided for policymakers concerned with the research and development of machine-based learning systems. The research method adopted emphasized iterative refinement of knowledge in response to actual experience; i.e., a machine's knowledge was acquired initially from a human who provided enough concepts, constraints, and problem-solving heuristics to define some minimal level of performance. Sixty-two references are listed. (Author/FM)

KNOWLEDGE ACQUISITION, KNOWLEDGE PROGRAMMING, AND KNOWLEDGE REFINEMENT

PREPARED UNDER A GRANT FROM THE NATIONAL SCIENCE FOUNDATION

FREDERICK HAYES-ROTH, PHILIP KLAHR,
DAVID J. MOSTOW

R-2540-NSF
MAY 1980

Rand
SANTA MONICA, CA. 90406

PREFACE

This report describes the principal findings and recommendations of a two-year Rand research project on machine-aided knowledge acquisition, supported by a grant from the Intelligent Systems Program, Mathematical and Computer Sciences Division, National Science Foundation. It discusses the transfer of expertise from humans to machines, as well as the functions of planning, debugging, knowledge refinement, and autonomous machine learning. Throughout, the authors attempt to explain the relative advantages of humans and machines in the building of intelligent systems. This viewpoint motivates an approach to hybrid systems in which humans provide initial knowledge to machines and cooperate with them in the iterative evaluation and refinement of the knowledge-based system. The report provides background and guidance for policymakers concerned with the research and development of machine-based learning systems, and it should be of interest to computer scientists, cognitive psychologists, and others interested in machine intelligence.

During this project, the research staff explored many alternative approaches to machine-aided learning and concluded that the most promising avenue for future work is to translate human expertise into initial intelligent programs and then to refine the implementations iteratively as indicated by experience. This approach blends machine capabilities with human expertise in the generation and iterative improvement of intelligent systems.

The report explains why the authors have chosen to explore this problem, describes how they are pursuing their goals, and discusses the progress to date. Because the stated task is very difficult, the surface has only been scratched; much remains to be done.

Several computer programs have been developed in the course of this project by various combinations of the authors or their collaborators. Mostow has implemented the knowledge-programming system as part of his dissertation research supervised by Hayes-Roth. Hayes-Roth and Klahr have implemented three different programs for representing and applying the knowledge of the hearts domain. While these programs actually play the game moderately well, their interest for us lies in their alternative approaches to knowledge representation and control. Lenat has implemented in his program RLL some of the ideas for restructuring knowledge discussed in Lenat, Hayes-Roth, and Klahr (1979a & 1979b). All of these programs should be considered fundamentally as experimental and ephemeral; they serve primarily as "breadboards" for testing our ideas. In the future, they may give rise to additional experiments or more permanent and extensive applied systems.

David J. Mostow is a doctoral candidate in the Department of Computer Science, Carnegie-Mellon University, and a consultant to The Rand Corporation.

SUMMARY

The scientist who wants to understand and construct learning systems must first define "learning" and then develop a research method that eventually produces learning mechanisms. In contrast to conventional views of learning that focus on problems of feature selection, combination, and weighting, the definition we have used in this study emphasizes a continual growth and refinement of knowledge. From this point of view, a system learns by increasing the scope, precision, validity, and power of its current knowledge. That knowledge includes descriptions and models of the environment, as well as planning and problem-solving techniques for achieving goals. Learning requires the system to augment, generalize, specialize, validate, and exploit the knowledge it possesses at any point in time.

The research method adopted here emphasizes iterative refinement of knowledge in response to actual experience. In this approach, a machine's knowledge is acquired initially from a human, who provides enough concepts, constraints, and problem-solving heuristics to define some minimal level of performance. We use semiautomatic methods to convert the initial knowledge into a working program. Then, when the initial program executes, we observe the resulting behaviors to diagnose problems and design knowledge refinements. Although this refinement process currently requires significant human contributions, we have formulated methods that may make it possible to reduce or eliminate much of that involvement.

This report explains the cyclic roles of understanding, planning, and evaluation during the development and extension of knowledge. A prerequisite for learning is an initial base of knowledge. Most learning results from the constructive application, evaluation, and refinement of that knowledge. By predicting probable consequences of our actions and noticing unfulfilled expectations, we can isolate and often quickly remedy weaknesses in existing knowledge. In this framework, a learning system must use existing knowledge to plan reasonable courses of action, carry those plans out, and then diagnose weaknesses that explain observed failures or unexpected successes. The processes of planning, acting, and evaluation cooperate to produce new knowledge by refining and extending prior knowledge.

The suggested approach is illustrated by the following simple example: Let us suppose that an intelligent system has been designed to control a power generator. Its knowledge base describes the system's likely behaviors, sensor readings, and control mechanisms. These behavioral specifications then relate assumed situation conditions to optional control actions and their expected effects. For example, the knowledge base might describe the relationships among several valves, pumps, pipes, temperature gauges, and control switches. Let us assume that rising temperatures produce some undesirable situation D. A plan to compensate for rising temperatures indicated by sensor A might dictate turning off valve B on pipe C. This plan would be associated with a rationalization, i.e., a proof or informal argument, establishing that when A indicates rising temperature, closing B on C will counteract the undesirable consequence D. The proof might also establish additional expectations about corroborative sensor readings or side-effects produced by the B closure. Let us next suppose that, subsequently, D actually occurs. We can now

compare the assumptions and deductions in the rationalization with actual observations of the current situation. We may find that all the assumptions remain true, but some deductions are falsified by data. In this case, we can identify the faulty inference rules as those whose premises are true but whose conclusions are false. Alternatively, we may find that some of the expected or presupposed conditions do not obtain. In this case, too, we can identify the source of the faulty expectation. Once it has been identified, the faulty rule becomes the subject of refinement.

In refining a faulty inferential rule, a variety of heuristics may be used. Each potential change to existing knowledge reflects a new hypothesis about the behavior of the modeled system or the effects of actions upon that system. These hypotheses constitute the intelligent system's new knowledge. To illustrate any of these learning heuristics requires an actual comparison between expectations and observations. But let us suppose that the plan for preventing D reasoned as follows: Closing B prevents water from reaching E (B-E rule), which, in turn, prevents vapor from saturating F (E-F rule); and only a saturated F could cause D to occur (F-D rule). Now a variety of diagnoses and refinements might arise. To pursue one of these, suppose that F does not saturate but D occurs; we should reject the current F-D rule. To refine this rule we would need to specify more accurately those situations in which the predicted F-D relationships occur. We might then use any of several available techniques for constructing descriptions capable of distinguishing the current situation from earlier ones in which the F-D relationship behaved as predicted. The new discriminating specification would yield a new hypothetical rule such as, When condition X occurs, only saturating F causes D.

Alternatively, we may find a source of improved knowledge in the proof that underlies the plan. This proof might introduce some unproved and invalid assumptions. In the current example, the belief in the effectiveness of closing B for precluding D might rest on the assumption that saturating F causes it to short-circuit and, implicitly, that *nothing else could*. In practice, many assumptions of this sort enter into planning quite implicitly. But in this case, a short-circuited F unit in the presence of nonsaturation reveals the faulty assumption directly. Many more illustrations and examples of the diagnosis and refinement process are described in this report.

Some of the ideas described here have been implemented in working computer programs that are described in detail elsewhere; however, in this study we have concentrated primarily on improved problem formulations, concept developments, and "hand-simulations." We anticipate that long-term research will be required to implement working computer programs that can perform all the functions this research paradigm suggests.

ACKNOWLEDGMENTS

We gratefully acknowledge the substantive contributions of our Rand, Stanford, and Carnegie-Mellon colleagues to this work. John Burge collaborated with us on learning heuristics. Stanley Rosenschein made frequent contributions to our efforts on knowledge representation. Douglas Lenat assisted in the development of knowledge representations, heuristics, and procedures for cognitive economy. As advisors to Jack Mostow, both Allen Newell and Jaime Carbonell contributed to our understanding of operationalization. This report, however, represents only the authors' attitudes and not the opinions of others with whom we have collaborated.

CONTENTS

I. BACKGROUND: MACHINE-AIDED KNOWLEDGE ACQUISITION IN PERSPECTIVE

The processes of knowledge acquisition and learning have fascinated people throughout much of history. In this century, the growth of behavioral and information sciences has stimulated various forms of basic and applied learning research. Behavioral psychology, for example, has made impressive gains in developing practical procedures for improved training of humans and animals. This type of work focuses primarily on the nature and the appropriate timing of contingent reinforcements. The essential finding has been that a reward received soon after some desirable behavior occurs increases the chance that the behavior will recur. Cognitive psychology, on the other hand, emphasizes the ideas and concepts governing ordinary thought. For example, researchers in this field attempt to explain how humans induce common category concepts such as "dog" or "criminal" from examples. Researchers in the relatively new field of artificial intelligence (AI) have addressed both of these kinds of learning problems. However, their emphasis on machine learning imposes demanding constraints on potential theories. For AI purposes, a theory of learning must lead to a computer program that exhibits improved performance over time.

This report describes only one of many approaches to machine learning. Thus, we shall first review briefly the related AI research that forms the current scientific context. The several approaches that have developed over time have emphasized, in order, adaptively adjusting feature weights; generalizing examples of categories, transformations, and more general procedures; using heuristics to synthesize new concepts; and directly transferring human knowledge to computers.

Early AI work studied adaptive learning schemes that could adjust control parameters to correlate the machine's output with a desired standard. In this sense, the early learning devices acted somewhat like adaptive control devices. The Perceptron (Minsky & Papert, 1969), for example, was a pattern-recognition device that classified test patterns by computing weighted sums of feature-detector outputs. When the sum exceeded some threshold, the response indicated corresponding class membership; if the sum fell below the threshold, the pattern was rejected from the class. When an incorrect decision occurred, a learning algorithm prescribed how to adjust the feature weights. Because the machine adjusted its weighting factors to accommodate its training experiences, this kind of learning might be considered the first of many subsequent paradigms for "learning by example." (A different application of a similar technique addressed tactics in the game of checkers (Samuel, 1963).)

Over time, AI researchers moved increasingly toward a belief that intelligent behavior requires substantial world knowledge. Most intelligent tasks require specific feature detectors, complex descriptions of patterns and structures, and correspondingly complicated procedures for comparing one description with another. Without these, very few human capabilities could be simulated. Soon after the development of behavioral learning theories and devices like the Perceptron, scientists began to point out the need for these more complex mechanisms. Work in machine-vision programs, for example, established the need for specialized detectors for edges, corners, and intersections and sophisticated procedures for following and interpreting their connections in scenes. These kinds of internal structures lay outside the scope of the earlier learning frameworks. A number of AI researchers then developed improved methods for learning by example that could generalize rules from arbitrarily complex structural descriptions (Buchanan et al., 1969; Buchanan & Mitchell, 1978; Hayes-Roth, 1974; Hayes-Roth, 1976a; Hayes-Roth, 1976b; Hayes-Roth, 1977; Hayes-Roth, 1978b; Hayes-Roth & Burge, 1976; Hayes-Roth & McDermott, 1976; Hayes-Roth & McDermott, 1978; Mitchell, 1977; Soloway & Riseman, 1977; Vere, 1978a; Vere, 1978b; Winston, 1975). These procedures use initially provided feature detectors as well as "structural" or "relational" connections to describe each example of a given class. Then by partial-matching the descriptions of many examples, common subdescriptions emerge as candidate general rules (Hayes-Roth, 1978a). This methodology has supported machine induction of transformational grammar rules (Hayes-Roth & McDermott, 1978), chemical reaction rules (Buchanan et al., 1969; Buchanan & Mitchell, 1978), and simple robot plans (Vere, 1978a), among others.

The basic limitation of this more recent work derives from its "subtractive" approach to learning. The learning programs devised under this approach produce new rules by detecting which of the currently known features and relations appear jointly in each example. By assumption, each example reflects criterial features as well as some irrelevant features peculiar to the specific example. Learning by example, in this context, simply requires subtracting the irrelevant features in each case. While this approach can be very useful for practical problems in pattern recognition and data interpretation, it provides little insight into the discovery of new features or new functions for performing a task.

Two recent research projects have shed some light on such discovery problems. Exemplary-programming research at Rand (Waterman, 1978b; Faught et al., 1980) has investigated the problem of inferring programs capable of recreating the interactions between a human and a machine engaged in a task (see also Biermann & Krishnaswamy, 1976). The creation of a program from a human/machine dialogue requires methods that are more constructive or "synthetic" than other learning-by-example tasks. The two chief problems concern interpretation of the example behaviors and the subsequent regeneration of corresponding behavior in new contexts. Interpreting an arbitrary human/machine interaction appears to require a variety of sources of knowledge, including sources for explaining (1) the meaning of special typed symbols, (2) the state of various systems employed during the session, (3) the semantics of computer system outputs, (4) the goal of the person performing the task, and (5) the problem-solving procedure that person apparently applied. Each of these types of knowledge contributes to understanding both why and how the person and the machine cooperated to solve the task. To construct a generalized program that can replace the person in such tasks, we must convert this passive understanding of the task's purpose and solution methods into effective procedures. These procedures, if truly general, must accomplish the same effects although various situational characteristics will differ from the initially observed data. This requires several types of knowledge in addition to that previously noted: knowledge concerning system control and interactions; knowledge of planning and problem-solving; and knowledge of programming methods.

Another recent project that illuminates the synthetic nature of learning attempted to simulate the discovery process in mathematics (Lenat, 1976; Lenat, 1977a; Lenat, 1977b; Lenat & Harris, 1978). This project employed two types of knowledge to induce new concepts of elementary set theory. The first type of knowledge consisted of a variety of mathematical concepts, such as sets, lists, equalities, and functions. Over time, the program's conceptual knowledge grew as new concepts were created from existing ones. The methods for discovering new concepts constituted the second type of knowledge. A few hundred rules called "discovery heuristics" modified existing concepts to produce new ones. For example, several heuristics formulated new concepts by "generalizing" old ones. Although the program knew at the outset the concepts of a *list* (defined as "an ordered collection of elements") and *length*, it conjectured for itself a new concept that generalized these notions to produce the concept of "length of a list of identical elements." In this way, it produced the concept of *unary numbers*, that is, a list of *n* tick marks meaning the number *n*. Other rules formed new concepts by specializing existing concepts, by searching for examples of newly conjectured concepts, or by forming new mathematical functions with arbitrary attributes (e.g., by restricting binary functions that applied only to cases where the first and second arguments were equal). In this way, several insightful and interesting developments of mathematical history were retraced in a few hours of computer time.

The last type of machine learning we must mention might be called "transfer of expertise" (Anderson, 1977; Balzer et al., 1977; Davis et al., 1977; Davis, 1977; Davis, 1978; Hayes-Roth et al., 1978; Heidorn, 1976; Heidorn, 1974; Mostow & Hayes-Roth, 1979a; Mostow & Hayes-Roth, 1979b; Samuel, 1963; Waterman, 1978a; Waterman et al., 1979; Barr et al., 1979). Work in this area has aimed toward constructing intelligent systems according to heuristic techniques prescribed by human experts (Feigenbaum, 1977). The major obstacle to implementing intelligent programs described in this way arises from the need to translate human knowledge into computable formalisms. Several research projects have demonstrated the viability of using English-like, rule-based languages (Anderson & Gillogly, 1976; Davis & King, 1976; Shortliffe, 1976; Waterman & Hayes-Roth, 1978; Waterman et al., 1979) that enable humans to express their knowledge in rules of the form, If there is a drilling site whose iron content exceeds 12 ppm and whose location is within 12 miles of an oil field, then the probability of a moderate iron deposit is high. While this particular rule is fanciful, a number of expert systems have been created for problems as diverse as infectious blood-disease therapy (Shortliffe, 1976), artificial respirator maintenance (Fagan, 1978), internal medicine (Pople, 1975 & 1977), and geological prospecting (Duda et al., 1978). The major lines of continuing effort in this context aim (1) to develop improved high-level languages for such rule-based programming (Waterman & Hayes-Roth, 1978; Waterman et al., 1979) and (2) to assist in the construction and maintenance of large sets of rules by developing metaknowledge—knowledge about the likely and appropriate kinds of knowledge that should enter the data base (Davis, 1978; Davis, 1979; Stefik, 1979).

The principal concepts of previous learning research that we have discussed are summarized in Table 1. The table also gives rough definitions of these terms as well as brief descriptions of the mechanisms AI researchers have proposed for accomplishing various types of learning.

Our current research attempts to integrate and extend the best aspects of these alternative approaches to machine learning and knowledge acquisition. Specifical-

Table 1

PREVIOUS LEARNING CONCEPTS AND MECHANISMS

Concept	Traditional Meaning	Mechanism
Learning	Changing behavior to improve performance	Rote memorization and conditioning to contingent reinforcement.
Adaptive learning procedures	Changing behavior output to more closely approximate desired standard.	Adjust association weights connecting stimulus features to outputs by feedback (positive/negative).
Learning by example	Inferring a general classification rule from training sets.	Propose any Boolean combination of features consistent with the examples as the classifying rule.
Exemplary programming	Inferring a general procedure from sample human/machine dialogue.	Infer the human's purpose that motivates the dialogue, interpret the dependencies between machine and human inputs, then create a program to mimic the human.
Concept discovery	Inferring the existence and definition of a general class.	Detect that several distinct things or events share some features, find a common description, and propose it as a concept definition; or heuristically modify a prior concept definition, conjecture the concept's validity, and find examples of it.
Transfer of expertise	Supplying human problem-solving knowledge to a machine.	Program corresponding procedures directly; or express the knowledge within a narrow formalism suitable for machine interpretation; or express the knowledge in a high-level problem-solving language (e.g., a rule-based system).
Knowledge acquisition	Incremental addition of knowledge to an intelligent system.	Formulate a representation for a type of knowledge the system uses, use any mechanism to identify new units of knowledge, and add these to the knowledge base.

ly, we believe the following contributory factors to learning can and should be accommodated in a single system:

1. Contingent reinforcement—when behavior produces undesirable consequences, the knowledge responsible should be altered, and tendencies that produce desirable outcomes should be strengthened.
2. Learning by example—systems should benefit from and generalize their experiences.
3. Knowledge as the source of power—a learning system should acquire, manipulate, and apply knowledge in the pursuit of increased capabilities.
4. The understanding of goals and the capability for planning to achieve them.
5. Acceptance of human advice and knowledge about the task.

Integrating these five capabilities into a system will require significant effort, and only a small proportion of the needed programs have been implemented to date. On the other hand, we now understand many of the residual problems clearly. Section II explains our formulation of and approach to the problem of understanding and assimilating knowledge about an arbitrary task. The concepts and mechanisms that characterize our approach are summarized in Table 2. Knowledge is acquired from humans, used to develop plans for achieving goals, and finally converted into executable programs. Section III discusses an iterative cycle of planning, acting, and evaluation that relates initial knowledge tu specific behaviors and finally to new or modified concepts. After relating observed effects to specific problematic components of plans and, in turn, attributing these to erroneous elements of domain knowledge, tne system diagnoses deficiencies and conjectures new knowledge elements. Such knowledge refinements engender a new cycle of plans, acts, observed effects, and inductions. Learning, in this paradigm, is equivalent to the iterative improvement of performance arising from discoveries made while implementing and refining knowledge.

Table 2

LEARNING CONCEPTS AND MECHANISMS IN THE CURRENT APPROACH

Concept	Meaning	Mechanism
Human/machine cooperation	Human and machine cooperate to build intelligent systems and improve them over time.	Human provides initial advice on task knowledge and problem-solving methods; human aids in converting the advice into working programs and in diagnosing and refining knowledge.
Advice taking	Humans define domain concepts, specify behavioral constraints, and suggest problem-solving methods.	Develop a knowledge representation, express the knowledge in this form, and integrage it into a working program semiautomatically.
Goal-directed planning	Development of a plan for achieving the expressed goals that uses suggested methods and satisfies expressed constraints.	Work backward from goals to sufficient conditions and actions by deductively pursuing logical, heuristic, and instrumental transformations. These transformations symbolically manipulate the domain concepts, constraints, and problem-solving methods to produce an effective procedure. The planning process establishes assumptions and expectations about plan-related situations and effects.
Contingent reinforcement	Strengthening of rewarded behavioral tendencies and weakening of others.	Reinforce knowledge contributing to favorable outcomes; diagnose and refine knowledge causing failures.
Learning by example	Use of actual situations and results to trigger learning.	Compare expected outcomes to actual outcomes and assumed conditions to observed conditions in order to diagnose fallacious planning knowledge and to suggest knowledge refinements.
Knowledge refinement	Amendment or extension of knowledge in response to behavioral feedback.	Adjust the conditions assumed necessary or sufficient for an action to produce an effect; generalize or further specify the expectations associated with an action.

II. ADVICE-TAKING AND KNOWLEDGE PROGRAMMING

TWO PARADIGMS FOR DEVELOPING EXPERT SYSTEMS

Many of the recent successful applications of AI have shown the power of implementing, more or less directly, the heuristic rules of human experts. In tasks requiring only one or two types of inferential procedure, such as interpreting symptoms and test results in medical diagnosis (Shortliffe, 1976), nearly all domain knowledge can conform to a few generic forms or representations. Many applications of this type have adopted an *if-then* rule format for expressing the causal and inferential relationships (Feigenbaum et al., 1971; Waterman et al., 1979). To derive implications of known facts, the system applies the rules to any data that satisfy the rule antecedents (the *if* component), and the corresponding rule consequents become derived facts. The system infers likely causes of observed symptoms by reasoning deductively from observations to plausible causes. When the situation data match rule consequents (the *then* components), the system hypothesizes that the associated antecedent conditions may also be true. In this way, the system reasons backward from effects to likely causes.

These systems succeed, in part, because they use constrained rule formalisms and perform only one or two specialized kinds of inference. These constraints enable program designers to provide naturalistic languages in which nonprogrammers can conveniently express their knowledge. Such English-like languages make it easy for experts in various domains to create large and powerful rule sets by communicating directly with the computer. Because of their specialization, these systems can also help in checking the consistency and completeness of the human rule sets.

The simplicity of these systems, of course, means that they lack the capabilities needed for solving most types of problems. Although many domains have problems similar to the medical diagnosis problem, most intelligent systems need to perform a wider variety of actions than interpreting symptoms in terms of their underlying causes. If we consider such interpretation tasks as a special kind of perceptual process, we can easily see that intelligent behavior involves more than interpretation. Usually, we think of perception, planning, adaptive control, and knowledge acquisition as essential components of intelligence. We are presently unable to formulate all of these activities in terms of one uniform type of representation that requires only a small number of related inference methods.

Thus, although a human expert may know exactly what intelligent behavior requires in some new domain of interest, the current state of the art requires that a human programmer design and implement a unique system for most new tasks. Weeks or months after the initial transfer of knowledge from the expert to the programmer, a program emerges ready to run. Figure 1 illustrates this typical process.

Of course, when the program finally runs, it typically produces a variety of unexpected results. At this point, both the expert and the programmer discover that the original knowledge apparently underspecified the program, because a variety of situations produce unanticipated effects. Four types of problems explain

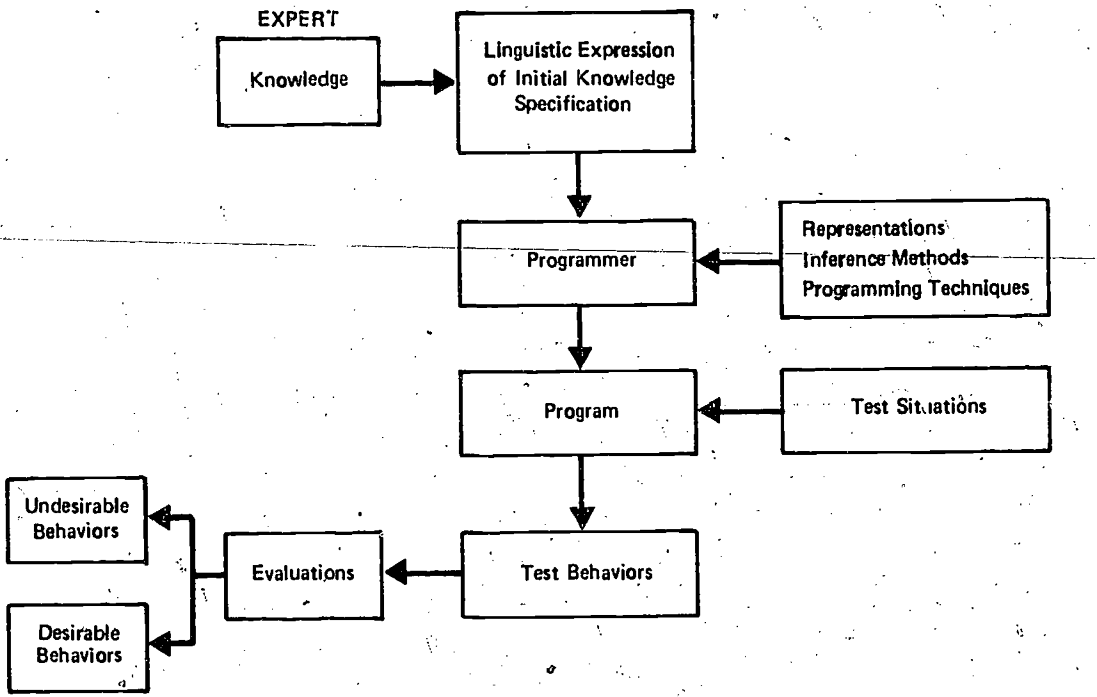

Fig. 1—Expert system development and testing

most of the behavioral deficiencies: (1) The expert neglected to express rules to cover all of the special cases that arise; (2) the expert's rules did not produce correct conclusions, because they made erroneous assumptions; (3) although the programmer's implementation decisions were consistent with ambiguities in the original specifications, they generated undesirable behaviors; or (4) the programmer overlooked or incorrectly implemented some of the expert's advice.

Observations of undesirable behaviors motivate a variety of discoveries and changes: New knowledge arises from efforts to handle additional special cases; the expert modifies his initial knowledge to correct the errors in it; the expert refines his initial rules to resolve problematic ambiguities; the programmer modifies his inference methods or associated program code so that the program behaves as it should. Unfortunately, all of these changes require programmer intervention, and most of them take significant time and effort.

Iterative refinements generally cause programs to become progressively more obtuse in their knowledge representations and control structures. This, in turn, makes it increasingly difficult for the expert to manage or comprehend his knowledge-based system. Ordinarily, the costs of programming and refinement are onerous; few programs ever satisfy their designers, because design goals continually evolve as experience reveals additional system shortcomings.

This analysis suggests an alternative paradigm for the programming and iterative refinement of intelligent systems. The principal components of this paradigm are shown in Fig. 2.

This paradigm views the programming problem primarily as one of translating expert advice into an operational program, and the iterative improvement problem

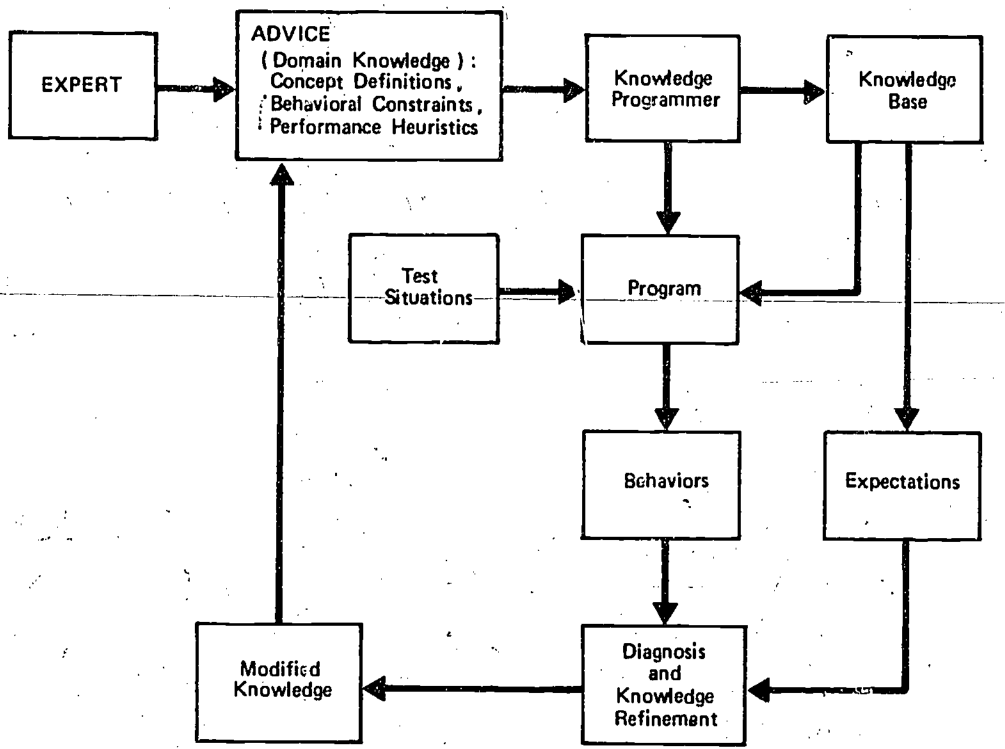

Fig. 2—Knowledge programmiug and knowledge refinement

as one of diagnosing program behavior to modify those elements of knowledge that produce undesired behaviors. This proposed scheme emphasizes the problems of understanding high-level advice, converting it into effective behavior, and, inevitably, changing the knowledge and reiterating the cycle. These problems are referred to as *knowledge acquisition, knowledge programming,* and *knowledge refinement,* respectively.[1] For some time to come, all of these processes will require some human participation. Thus, throughout this report we will describe semiautomatic procedures for performing these functions.[2]

In the remainder of this section, we explain the primary problems and proposed methods for the first two processes, i.e., acquiring the expert's knowledge by understanding advice and converting this advice into executable programs. These are the tasks of the *knowledge programmer.* In the process of knowledge programming, the knowledge programmer develops plans and procedures used by the resulting program. These plans create expectations concerning the way the program should behave. Contrasts between observed and expected behavior stimulate highly con-

[1] Knowledge acquisition, in our paradigm, refers to the transfer of expertise from a human expert to a machine. The machine *acquires* the human's knowledge in the form of concepts and heuristics. When the machine extends its initial knowledge by various learning methods, we refer to this as *knowledge refinement.* Different researchers might apply the term *knowledge acquisition* to varying aspects of these processes.

[2] We do not know if machine learning techniques will ever achieve sufficient levels of success to obviate the role of humans in such efforts. Thus, we see machines more as calculating aids to humans than as standalone investigators of complex domains. For the foreseeable future, at least, humans will play a major role in guiding deductive processes that the machines execute more rapidly and systematically than would otherwise be possible.

strained searches for underlying deficiencies in the knowledge base. These deficiencies, in turn, suggest knowledge refinements. The refinement processes of diagnosis and knowledge modification are described in Sec. III.

CONVERTING KNOWLEDGE INTO AN EXECUTABLE PROGRAM

We believe that a very large class of intelligent systems can be specified quite easily. A behavioral description would include constraints on permissible actions as well as prescriptive methods for attacking problems in the domain. Our approach rests on the central idea of expressing these behavioral constraints and heuristics in terms of natural domain concepts. A description of mathematical discovery methods, for instance, should employ terms familiar to a mathematician. When we ask a mathematician for advice abo his problem, such as *what to do* or *when and how to do it*, we should allow him to talk in his own terms. Asking him to express his knowledge in terms familiar to computer programmers forces him to translate mathematics into programming. On the other hand, asking a programmer to bridge the gap between the mathematician and the computer requires him to translate between cultures—he must first comprehend much of the field of mathematics and then map its concepts and methods into his own repertoire of computer capabilities.

To obviate the need for cross-cultural translation, we are developing systems that will accept an expert's advice expressed in familiar domain concepts. To understand the advice, we need to know the meaning of each constituent concept, and we must transform higher-level advice into actual procedures that the computer can perform. Our approach aims to assimilate individual concepts as terms with formally defined properties. Then, we attempt to understand advice (e.g., constraints among elements, or heuristic rules for goal-directed actions) as specific compositions of the constituent terms. At present, much of this overall process is understood, and some of it has actually been accomplished by our computer programs. We will sketch the primary features of this approach in the following paragraphs. (Detailed technical descriptions are given in Hayes-Roth et al., 1978; Mostow & Hayes-Roth, 1979a; Mostow & Hayes-Roth, 1979b.)

From the general perspective, *knowledge programming converts advice expressed in some naturalistic syntax into actions in the task environment.* This requires several processes: *parsing* the advice into syntactic structures; *interpreting* these structures by converting them to meaningful semantic representations; *operationalizing* the meaning structures by converting them into effective, executable expressions; *integrating* multiple pieces of advice into a coherent set of procedures; and, finally, *applying* these procedures in actual situations to generate actions.[3] These operations use a knowledge base that stores individual domain concepts and their interrelationships (see Fig. 3). Assimilating each new piece of advice

[3] While all five of these subtasks are difficult, we have focused our efforts on the problem of operationalization. Parsing and interpretation are fairly well understood as a result of the attention they have received in natural-language-understanding projects. Operationalization is a new topic of study. In the new paradigm, emphasizing rapid implementation of expert knowledge and rapid reimplementation of modified knowledge, operationalization plays a crucial role. Integration is also a very important and difficult problem. Unfortunately, we have not addressed this problem in a substantial way during the course of this research project. A neat solution to the integration problem would yield a single, comprehensive program that could be applied simply, as if it were a typical computer procedure.

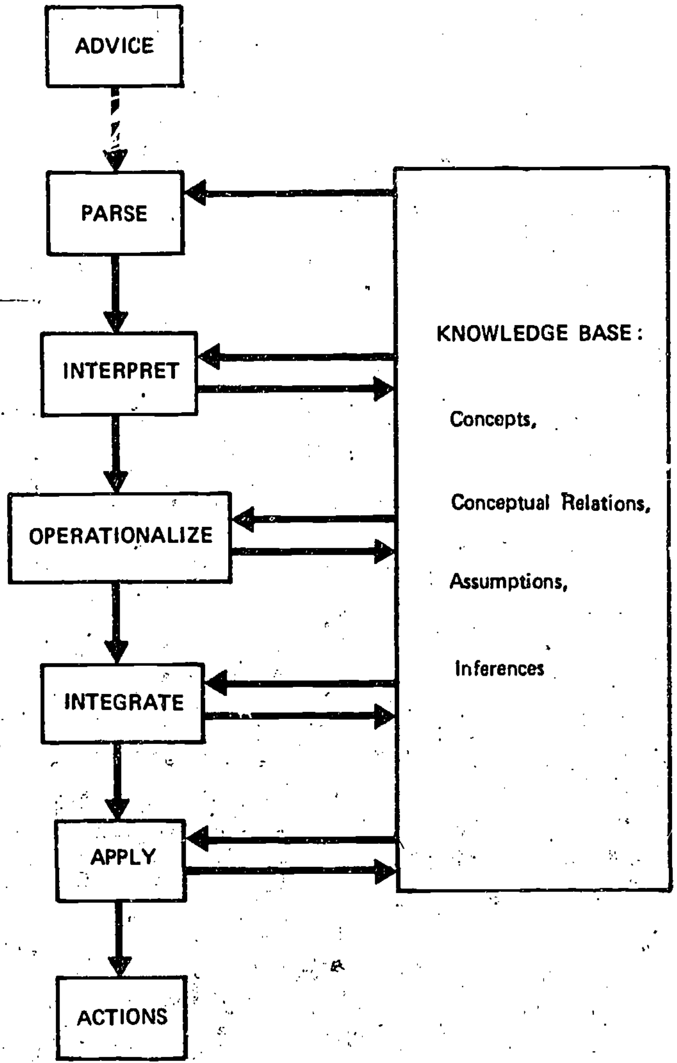

Fig. 3—Information flow in knowledge programming

requires the knowledge programmer to appreciate how the advice relates to other elements of the knowledge base. Thus, the knowledge base both feeds the assimilation processes and represents the incremental additions they produce.

Several different tasks we have explored in this paradigm have exhibited remarkably similar properties. Each task employs a small set of domain-specific concepts, a few constraints or rules, and an open-ended set of prescriptive heuristic methods. Such problem domains include music composition, legal reasoning, tactical planning, and game playing. We have found that a familiar card game, hearts, provides a good basis for illustrating the major points of this research (see also Balzer, 1966). The problem represented by the game of hearts is discussed below.

EXAMPLES OF OPERATIONALIZATION

What does it take to specify a program that will simply play hearts in accordance with the rules?[4] The published rules of the game define mandatory behavioral constraints. A program that behaves in accordance with these rules will play a legal, albeit poor, game. A few simple rules will illustrate the nature of this advice-taking problem.

Players rule. The game is played by four players.[5]

Players sequence rule. During a trick, players play in clockwise order around the table.

Trick rule. A trick is a sequence in which each player plays one card.

Trick leader rule. The first person to play in the first trick is the one who has the two of clubs. The first player plays the two of clubs. In other tricks, the winner of the preceding trick plays first.

Follow suit rule. Each player, if possible, must play a card in the suit of the first card played in the trick.

Win trick rule. In each trick, the player who plays the highest card in the suit led wins the trick.

The processes of parsing and interpreting these rules would generate knowledge-base elements of the following sorts:[6]

Players rule.
Players = {p1, p2, p3, p4}

Players sequence rule.
if p1 has just played in a trick and p2 has not played
  in the same trick then p2 plays next;
if p2 has just played in a trick and p3 has not played
  in the same trick then p3 plays next;
if p3 has just played in a trick and p4 has not played
  in the same trick then p4 plays next;
if p4 has just played in a trick and p1 has not played
  in the same trick then p1 plays next;

Trick rule.
if t is a trick
then q1 plays c1 during t

[4] The reader who is concerned with the concept rather than the technical details of this problem can skim the technical material that follows without seriously affecting the continuity of the discussion.

[5] For simplicity, we ignore variations on this rule (for example, one variation allows a three-player game).

[6] To avoid unnecessary formality and technicality, we have expressed propositions and conceptual statements in terms of simple English equivalents. Readers interested in our LISP-based representations may refer to the technical reports cited previously.

```
    followed by q2 plays c2 during t
    followed by q3 plays c3 during t
    followed by q4 plays c4 during t
    and {q1,q2,q3,q4} = players
    and c1, c2, c3, c4 are elements of cards.
```

Trick leader rule.

```
if and only if player p has card two of clubs
then player p plays first in the first trick
    and player p plays the card two of clubs in the first trick;

if and only if player p wins a trick t
then player p plays first in the trick t' following t;
```

Follow suit rule.

```
if the first card c played in a trick t is of suit s
    and player p before playing in trick t has some cards in suit s
then player p plays some card d in suit s during trick t;
```

Win trick rule.

```
if and only if
    the card c is the first card played in trick t
    and the suit of card c is s
    and the cards played in trick t are called C
    and C' is the subset of C whose suit is s
    and the highest valued element of C' is card d
    and card d was played by player p
then player p wins trick t;
```

The concept definitions supporting such interpretations are of two sorts. The first are domain-dependent definitions, such as *card, game, hand,* and *play.* For example, play would be defined as an action by a player that changes the location of a card from the hand of the player to the pot. The possible locations of cards include a player's hand, a player's pile, and the pot. In our work, we use a form of the lambda calculus (Church, 1941; Allen, 1978) for encoding these definitions. The second type of definition is used for representing domain-independent concepts, such as *set, subset, some, exists, sequence, if, then,* and *and.*

Let us suppose for the current example that only the preceding rules were specified, along with a minimal additional set prescribing how players receive their initial hands and how the winner of each trick takes the cards in the pot. Could we directly apply the knowledge to produce behavior? The answer is no, because the constraints recognize acceptable behavior without telling us how to generate it. That is, the constraints partially define *what* to do without explaining *how.* Discovering how to achieve such desired behavior requires us to *operationalize* the advice. In effect, we need to convert the advice concerning *what* into executable *methods.*

Even in this simple case, the search for effective methods to achieve the desired goal requires planning and problem-solving, which incidentally produce additional insights into the domain.

The general methods we have employed for operationalization may be summarized as goal-directed planning. We begin with a statement of desired behavior which, in this example, might be "player p plays a card c in the current trick t." The problem is to convert this goal expression into a procedure that achieves the goal and satisfies the constraints. Because this statement does not tell us how we may choose a satisfactory card, we regard it as an *ineffective expression*. We then attempt to transform this expression into a composite of individual subexpressions where each component represents an *effective expression;* that is, each corresponds either to a known value or an executable procedure that can produce a value. Several different problems need to be solved, and we use a variety of different methods.

In this simple example, we might attempt to transform the expression by adapting a general-purpose AI problem-solving method to this task. For example, we might attempt to adapt the general method of *generate-and-test* (Newell, 1969; Newell & Simon, 1972). To exploit this method, we need only find some generator that suggests each possible action and then apply some effective procedure for verifying that the suggested action satisfies the constraints. This approach would lead us to an operationalization such as the following:

```
Plan 1:
  consider each card c in turn;
  assume you were to play c;
  if you can prove that c would satisfy the constraints,
    then play c.
```

This plan has two interesting aspects that reflect some deep and recurring issues. First, formulating a plan of action without having the actual situation data in hand requires very general and abstract reasoning. When we do not know exactly which cards have been played and which cards are in a player's hand, for example, we have difficulty formulating any specific action. In an actual situation, however, we might find a similar judgment straightforward. For example, we could easily prove that playing the three of clubs is legal when the two of clubs has just been played. These two alternative tasks—forming general plans before situations unfold vs. planning on the spot—we distinguish as ordinary operationalization vs. *dynamic operationalization*. The first point then is that in most circumstances, dynamic operationalization seems both easier and more effective. Its primary disadvantage arises from the need to postpone planning until actions are required. This means that, at each point, the system cannot act until it has thought through the issues.

The second point concerns the *degree of effectiveness* we demand from an operationalization. The general plan proposed above may or may not be effective. It presupposes two capabilities: (1) generating each possible card in turn and (2) proving that the card satisfies the constraints. While in simple cases like the hearts task we can assuredly achieve the first capability, we can rarely establish valid proofs of such abstract propositions as in the second. The difficulty arises from the complete generality of the assertion to be proved, i.e., "c would satisfy the constraints." To certify Plan 1 as *effective*, we need a procedure that we know can prove such assertions. Thus, we might simply assume the use of a general theorem-proving procedure to perform this subtask. This procedure, in turn, may or may not be assuredly effective. The overall degree of effectiveness of Plan 1 would then depend on the theorem prover's own effectiveness and the eventual success or failure of its theorem-proving efforts.

The point we wish to make in this context concerns the degree of effectiveness we ascribe to an operationalization. Two kinds of uncertainties ordinarily preclude developing assuredly effective plans. First, most real-world tasks address inherently uncertain environments. In these tasks, the constraints and behavioral heuristics may not be strictly provable. Worse yet, in tasks that require both reward-seeking and risk-avoidance behaviors (e.g., playing hearts), knowledge usually prescribes simultaneously opposing, hence inconsistent, objectives. Second, plans themselves introduce a different type of uncertainty. This uncertainty, considered above, arises from the residual ineffectiveness or incompleteness of the operationalization. We have found that both kinds of uncertainties require heuristic solutions. That is, intelligent systems can reason informally to partially control uncertainty but they cannot eliminate it completely. Much of our research revolves around the nature of the heuristics both human experts and knowledge programmers can employ to control such uncertainty efficiently.

Because Plan 1 represents quite a "weak" or unspecific operationalization, we might reasonably seek a stronger, or more detailed solution. A more interesting plan for playing a card can be derived by adopting another general AI problem-solving approach. In this case, we view the problem from the perspective of the general *heuristic search method.* This method specifies that to reach a goal state G from an initial situation S, we should choose actions from some set A that successively change attributes of S until they satisfy G. To apply this general method to the problem at hand requires matching aspects of the given problem to components of the general approach. What elements of the current problem correspond to G, S, and A? The initial statement corresponds to G: Reach a state where player p plays a card c that satisfies all of the constraints. Situation S corresponds to a set of quite general assertions about what is known, such as that player p has cards C. The set of possible operations A is not immediately apparent.

To transform an initial state S into a goal state G, we employ three different kinds of transformations: logical, heuristic, and instrumental. Logical transformations convert an initial expression into a logically equivalent one. One common kind of logical transformation is a symbolic type of case analysis. To analyze an expression, we reexpress it as a set of alternatives, each of which rests on an additional, distinctive assumption characterizing that particular case. For example, we might reexpress the assertion that p plays card c as a disjunction of two cases: (1) p plays first in the trick and plays card c; or (2) some other player p' plays first in the trick, and later in the trick p plays card c. This transformation preserves the truth value of the initial expression, but, more importantly, it suggests one promising way to divide and conquer the initial problem.

Case analyses break a single general problem into separable subproblems and at the same time further characterize aspects of the task situation that bear directly upon constraint satisfaction. In Case 2, where p is assumed to play after the trick has begun, p must choose a card in the same suit as the first one played. In this example, the Case 2 assumption can key the knowledge programmer to apply the *follow suit* rule in subsequent operationalization of this path.

Heuristic transformations make plausible, if not necessarily valid, substitutions in expressions. For example, in developing a plan of play with attention to likely effects, we might transform the initial expression "after the trick is opened, p plays card c" into the two following cases: (3) "after the trick is opened, p plays card c, and c is the trick-winning card," or (4) "after the trick is opened, p plays a card whose suit is different from the suit of the first card played." While these cases are verifiably exclusive, they do not exhaust the possible situations that might arise. Hence, a transformation of this sort might be plausible but not equivalence-preserving. We have found many practical situations where such heuristic transformations can lead to clever, if somewhat incomplete, plans. In this illustration, for example, Case 3 leads to application of the *win trick* rule, further specifying the suit and value of card c. Case 4, on the other hand, suggests an operationalization that confronts the *follow suit* rule. The prerequisites for that rule in turn become additional conditions on this line of reasoning.

The third type of transformation to expressions models the result of instrumental action in the task environment. In the initial problem of generating a legal play, the only actions known to the system would be dealing, playing, leading, and winning a trick. Each of these corresponds to a transformation that will affect the description of any supposed game situation. In the analysis of Case 3 above, in addition to inferring requisite properties of the winning card c, we can also deduce that p moves cards from the pot to his pile. In this simple example, the only instrumental action that sheds much light on operationalizing the initial expression is the primitive action of playing a card, i.e., moving a card from a player's hand to the pot. Later, however, the actions of winning a trick and moving cards from the pot to players' piles will play significant roles in operationalizing behavioral heuristics suggested by experts.

The knowledge programmer represents a variety of general reasoning methods as transformations and applies these to convert high-level objectives into corresponding effective procedures. The reasoning methods exploited include case analysis, partial-matching between an expression and the description of a general method to guide attempts at adapting the general method to the specific problem, simplification of complex expressions, approximation of uncertain or combinatorial alternatives, and reformulation of an expression in terms of other known concepts. Each specific operationalization task requires some or all of these methods. For example, if the knowledge programmer adapted the heuristic search method to the task of playing a legal card, one resulting operational expression for "p plays card c and satisfies the constraints" would correspond to the following:

```
Plan 2:
  if p plays first in the trick
    and this is the first trick
    and p has the two of clubs
  then p plays the two of clubs,

  else if p plays first in the trick
    and p has a card c whose suit is s
  then p plays c,

  else if card c' was the first card played in the trick
    and the suit of c' is s
    and p has a card c whose suit is s
  then p plays c,
else if p has a card c
then p plays c.
```

Of course, because we have thus far considered only the mandatory constraints on behavior, no expertise has been included in this initial set of advice. In addition to necessary conditions on behavior, the kinds of heuristics we want to acquire directly from experts tell the program how it *should* behave. This goes beyond the notion of behavioral *acceptability* to the concept of *desirability*. As anyone familiar with law, music, hearts, and most other difficult tasks realizes, the bulk of human knowledge in these domains directly concerns such prescriptive heuristics.

OPERATIONALIZING STRATEGIES, TACTICS, AND PROCEDURES

In a game like hearts, where the real objective is to minimize winning tricks that contain point cards, expert advice concerns strategies, tactics, and procedures that can help reach this objective. The rules of the game reward some kinds of risk aversion and some kinds of risk-seeking behaviors. For example, a player can improve his or her (relative) score either by taking fewer points than the opponents or by taking all the points in a round. Thus, a very simple type of heuristic advice might be to "avoid taking points." We will consider this example briefly to convey the nature of the knowledge-programming problem it exemplifies. (A detailed technical discussion of this particular example appears in Mostow & Hayes-Roth, 1979b.)

Before proceeding with the example, however, we need to postulate a few more bits of knowledge. We will assume the knowledge programmer has assimilated the following facts: Any card that is in the suit of hearts has a point value of 1, and the queen of spades has a point value of 13. We assume also that the concept "take" has been defined to mean that a player winning a trick t takes all cards played in that trick, i.e., he moves them from the pot (cards played in the trick) to his pile. The concept "avoid an event x" is defined to mean "prevent event x" or "achieve not [x]." Using such basic definitions, our program has transformed the initial ineffective advice "avoid taking points" into an effective procedure. It generates a plan that recreates the typical high-level steps most people apparently follow, although it works through many more and lower-level steps than people consciously make.

Given the rules of the game and the advice to "avoid taking points," people reason roughly as follows: (1) Taking points means taking cards with point values; (2) the only way to take cards is to win a trick; (3) "avoid taking points" thus means not winning a trick; (4) this in turn means playing a card that is not the highest one; and (5) this suggests playing the lowest card in your hand.[7]

For the sake of brevity, we summarize the actual machine-aided derivation at the same high level as the introspective human analysis. First, the program logically transforms "avoid taking points" by substituting for the term "avoid" its literal definition. This produces an expression like "establish not [player p takes points]." Although the objects of "take" actions are cards and cards can have points, it is

[7] After a little thought, people often notice ways to improve this plan, but we shall reconsider those kinds of insights later when we discuss the role of plan evaluation as a source of knowledge refinement.

impossible to "take" points directly. The program reasons heuristically that "taking points" seems equivalent to "taking cards that have points." From this, the program notices that a sufficient condition for taking points is winning a trick in which some of the cards have points. To preclude this from happening, it reasons that negating any of the necessary conditions should do. It then produces a new expression that corresponds to "do not win a trick." It uses the constraints on trick winning to infer that the player wins only if he plays the highest card in the same suit as the card first played. Finally, it reasons instrumentally that this condition would not occur if he played a card of lower value. This type of plan leads directly to a corresponding procedure for applying the advice.

In a similar way, the program has been used with human assistance to produce plans for other kinds of advice in this game. For example, one useful heuristic for new players is to "flush the queen of spades," i.e., force another player to play it. The kind of reasoning the program uses to develop its plan is as follows: By substituting the definition of "flush," it infers that it needs to establish the condition "some player p must play the queen of spades." It uses its concept of "must" and the *follow suit* rule to infer that this objective requires that p have only one legal card to play—the queen of spades. This in turn entails either (1) p has only the queen of spades or (2) player q leads a spade, and p's only spade is the queen. It focuses on the second case and then develops a plan for how player q could force such a situation. In brief, it develops a plan for q to win a trick to take the lead. Then as long as q retains the lead, q continues to lead spades. As players familiar with the game will realize, this is an effective method for flushing the queen.[8]

INTEGRATION

We have done little thus far to address the question of integrating a variety of separate pieces of advice. This type of problem lends itself to two approaches. The first aims at an overall consistent integration, while the second presumes no such comprehensive integration is feasible. As in the preceding examples, operationalizing a single piece of advice often requires simultaneously satisfying numerous constraints. Such an approach to comprehensive integration fits the overall framework we have illustrated throughout this section. In the second type of approach, we presumably do not know the ways in which several pieces of advice interact or, worse yet, the ways in which independent pieces may contradict one another. This type of situation arises when we advise, for example, both "avoid taking points" and "take all the points" or "take at least one point if no one else has."

Our approach to integrating multiple pieces of advice takes two basic forms. First, we try to formulate independent recommendations that themselves may become the objects of *metaheuristics*. That is, we wish to accept advice about when, how, and why to combine or favor one heuristic over another. Second, we want to infer these dependencies by understanding why some heuristics produce undesirable results in actual situations. In such cases, we wish to eliminate the ambiguity by refining the initial heuristic to restrict its application to appropriate situations. This kind of refinement is discussed in Sec. III.

[8] Readers may also develop variations of this plan that seem superior. Such variations are discussed in more detail in Sec. III.

In sum, we have presented our paradigm for knowledge-programming and iterative reprogramming of intelligent systems. This section has focused on knowledge-programming processes, which are pertinent to both initial programming and recurrent reprogramming. The next section motivates and explains the reprogramming problem in more detail. We have briefly explained the kinds of advice we expect our systems to assimilate and a variety of methods for converting the advice into operational programs. In this process, we see that the knowledge programmer formulates plans that develop its initially vague concepts into effective procedures for accomplishing goals. These plans also play a major role in identifying weaknesses in knowledge that stimulate learning.

III. PLAN EVALUATION AND KNOWLEDGE REFINEMENT

BUGS REVEALED IN PROGRAM EXECUTION

To convert constraints and heuristics into action, the knowledge programmer develops a plan that integrates task-environment actions along with logical or heuristic inferences. In planning, the knowledge programmer reasons about the effects of the various transformations it employs. Some of its transformations preserve logical equivalence, while others introduce approximate or plausible reasoning. These latter kinds of transformations may introduce undesirable effects or "bugs." The second phase of intelligent system development is concerned with the *identification, diagnosis*, and *elimination* of such bugs.

We have developed a list of bugs that arise in knowledge programming. Some of these arise from omissions, errors, or ambiguity in the initial knowledge, while others are introduced by the knowledge-programming process. Table 3 summarizes these bugs. (Other AI researchers have considered bugs in problem-solving procedures, but these have little in common with those under consideration here; see, for example, Davis, 1978; Davis, 1979; Miller & Goldstein, 1976; Sussman, 1975; Sussmann & Stallman, 1975; Brown & Burton, 1975.)

Table 3

Bugs Arising from Knowledge Programming

Type of Problem	Source of Problem	Manifestation
1. Excess generality	Special cases overlooked	Good rule occasionally produces bad effects.
2. Excess specificity	Generality undetected.	Rules fail to cover enough cases.
3. Concept poverty	Useful relationship not detected and exploited.	Limited power and capability of system.
4. Invalid knowledge	Misstatement of facts or approximations.	Expert's expectations violated.
5. Ambiguous knowledge	Implicit dependencies not adequately articulated.	Conflicts arise in some situations about what is best to do.
6. Invalid reasoning	Programmer incorrectly transforms knowledge.	Knowledge programmer's expectations violated.
7. Inadequate integration	Dependencies among multiple pieces of advice incompletely integrated.	Rejected action alternatives actually satisfy more criteria than selected action does.
8. Limited horizon	Consequences of recent past or probable future events not exploited.	Judgmental logic seems static, not sensitive to changing or foreseeable situations.
9. Egocentricity	Little attention paid to probable meaning of others' actions.	No apparent adaptation of one's behavior to exploit knowledge of other's plans.

The nine problems listed in Table 3 span a large set of potential weaknesses in intelligent programs. For the sake of brevity, we will consider only one of these bugs in detail as an illustration of knowledge-refinement techniques. We have chosen invalid reasoning, because it illustrates many of the ideas that recur in knowledge refinement. After we have explained our approach to debugging reasoning problems, we will briefly characterize the approaches taken for the other kinds of problems as well.

DIAGNOSING AND FIXING A REASONING ERROR

We address here only a limited class of reasoning errors, namely those which manifest themselves as discrepancies between the observed outcomes of executing a plan and the expected outcomes. Here we are focusing on the expectations that the knowledge programmer generates as by-products of its operationalizations and integrations. The knowledge programmer acts as if it believes the transformations used to convert ineffective statements to specific procedures will produce results satisfying the original objectives. This belief applies, in turn, to each successive transformation applied during the planning process. However, the transformations may in fact yield procedures that do not always satisfy these expectations.

The approach we take to knowledge refinement in this type of problem begins with an attempt to analyze an unexpected event. Thus expectations motivate and trigger the knowledge-refinement process, as shown in Table 4. By analyzing the violated expectation, we identify both what went wrong and why. Then, we propose changes to the underlying knowledge to remedy the problem. The success of this method often depends on isolating missing, extraneous, or imprecise predicates used to restrict the time at which some action occurs. (This approach parallels that

Table 4

KNOWLEDGE-REFINEMENT APPROACH

Step	Source of Mechanism
1. Establish expectations	During knowledge programming, planning establishes plausible antecedents and consequences of actions; these beliefs represent expectations.
2. Trigger analysis	When an actual event violates an expectation, the reasoning behind the expectation is reanalyzed in light of observable data.
3. Locate faulty rules	A set of diagnostic rules debug the planning logic by contrasting the a priori beliefs with actual data. If a heuristic rule used by the plan assumes a false premise or entails a false conclusion, it is faulty.
4. Modify faulty rules	A set of learning rules suggest plausible fixes to the erroneous heuristic rule. These might alter its preconditions, assumptions, or expectations to keep it from producing the same faulty result in a subsequent situation.
5. Reimplement and test	Incorporate a modified heuristic rule into a new system by reinvoking the knowledge programmer. Verify that the rule eliminates the previous problem and test it in new situations.

discussed in Lakatos, 1976.) This will become clearer in the context of a concrete illustration.

As one example of the act-evaluate-refine process, consider what happens when the machine attempts to execute the previously developed plan to flush the queen of spades. The plan was, roughly, take the lead, then lead spades until a player is forced to play the queen. Suppose that this plan worked well in several games, but during one game a sequence like the following unexpectedly occurred: The machine player wins a trick. It then leads the jack of spades, and the other players follow suit. The queen is still held by one of the players. On the next trick, the machine chooses another spade to lead. This time, it has only two spades left, the four and the king, and it chooses arbitrarily to play the king. The next player plays the five, the one after him plays the queen, and the last plays the ten. The machine has just won a trick according to its plan, and it has even flushed the queen. Unfortunately, it has also taken 13 points, presumably a very undesirable outcome.

What might a person in the machine's situation do at this point? With apparently little effort, a person would recognize that the plan was buggy, because it achieved an undesirable result that was unexpected. Implicit in the plan was the notion that the player with the queen would be coerced into playing it and, presumably, winning the spade trick with it. In response to this insight, a human player would amend the plan appropriately. The fix in this case would require that when trying to flush the queen, a player must lead only spades below the queen.

Our learning methods capture the general logic behind this type of analysis. There are many chains of reasoning that might lead to the same proposed refinement as our hypothetical human produced. We will explain one type of argument that appears programmable.

Let us suppose that the machine (unlike a human) has no precise expectation regarding the queen-flushing plan. However, since it followed supposedly expert advice, it has a general expectation that bad consequences should not result. When, as in this case, undesirable results occur, the program tries to understand why it suffered such an outcome and how it could have prevented it.

The machine analyzes the last trick to infer cause-effect relations, based on its current knowledge. To take 13 points in the trick, it had to win the trick during which the queen was played. So it conjectures for itself some refined advice: Flush the queen of spades but do not win a trick in which the queen is played. Because this refined advice surpasses the original advice in quality, the machine has already improved its knowledge. On the other hand, this high-level advice requires operationalization if it is to be useful. However, our current knowledge programmer does correctly operationalize this advice by producing a plan corresponding to the following: Take the lead, then continue leading spades below the queen. Thus, this type of bug is eliminated by formulating a desired refinement directly in terms of a new high-level prescriptive heuristic. The refined heuristic, in turn, is implemented by the same knowledge-programming methods previously used for accepting advice from humans. (In some cases, as in this example, the refined heuristic can also be implemented simply by modifying the previous plan, as opposed to starting over from scratch.)

To continue our illustration, let us suppose that the machine begins to apply its refined plan. Because it knows that the plan has been refined to prevent it from taking the queen of spades itself, it notes this specific expectation in the knowledge base as a predicted consequence of the plan. In a new game, however, suppose it has the ten, jack, and king of spades. It wins a trick, then leads the ten. All players follow suit with lower cards, so the machine leads again with the jack. Again it wins the trick. At this point, its revised plan proscribes leading spades, so it plays a diamond. Another player wins the trick, and continues to lead spades. The machine is forced to play the king, and the player after it follows suit with the queen of spades.

Again, contrary to its specific expectation, it wins the trick and takes 13 points. Now it attempts to discover why its expectation was violated. It constructs a cause-effect model of the events leading to the disaster. In this model, it notes that at the time it played the king, it had no other choices. So apparently, by that time, only by keeping the other player from leading spades could it have prevented the disaster. Alternatively, it reviews events prior to that trick to see what, if anything, it did that contributed to creating a situation where no options existed. It notices that playing the ten and jack of spades earlier produced the state where it had only the king of spades. It notes that these actions were taken with the express intention of preventing it from taking the queen, but apparently they contributed directly to just that outcome.

It now proposes to itself another refinement. It should prevent a reoccurrence of this type of situation in the future. Its proposed advice: Do not lead low spades if you can be forced to play a spade higher than the queen.[1] This, in turn, leads to an operationalization that requires an estimation of the probable distributions of spades among players. While we have developed some methods for handling such probability functions, we have not yet implemented those needed for this particular problem. However, as persons knowledgeable in the game will notice, the proposed concept of a card that is "safe" vis-à-vis the opposing distributions is quite sophisticated. In fact, generalizations of this "safe spade" concept, such as "safe in suit x" or "safe with respect to all suits," play major roles in expert strategies.

As another example of knowledge refinement, consider again the plan developed in Sec. II to avoid taking points. That plan proposed playing the lowest possible card. Using this plan, the machine expects it will avoid taking points, but there are numerous ways that the plan leads to violated expectations, each of which reflects characteristics similar to those in the queen-flushing examples. For example, it may play its lowest card (a five, say) and still win a trick with points. This causes it to weaken its expectations (i.e., to associate some uncertainty with this predicted outcome). Pursuant to such a play, it may take another trick with its current lowest card (a ten, say), again with points. However, if it had played the ten before the five, it might have avoided winning the second trick, because in the second trick the five might have been lower than another player's card. Each of these problems gives rise to new attempts to refine both the expectations and the plan, in a manner similar to that previously described.

[1] This example has not actually been performed by a machine implementation. Before it could be implemented, several difficult issues would arise. Foremost among these, the diagnostic system would need to conjecture several alternative problems and solutions. Each of these proposed solutions would require, in turn, experimental testing through additional play. For example, the program might have hypothesized the remedy, Do not begin to flush the queen of spades if you cannot retain the lead. This heuristic seems beneficial, but we cannot be certain. Empirical validation of alternative heuristics seems unavoidable.

Our general knowledge-refinement strategy can be characterized simply, as shown in Fig. 4.

The contrast between expectations and actual outcomes focuses the learning system directly on specific problems. The system then attempts to find the flaws in its original causal model in light of the new data at hand. This in turn suggests additional conditions or new goals for knowledge programming.

The overall approach we have taken to this problem employs three basic elements: (1) proofs, (2) diagnostic rules, and (3) learning rules. While these steps have not actually been implemented on a computer, we have hand-simulated all of them. The knowledge programmer associates with each plan and its expectations a *proof* (or an informal rationale). The proof of a plan links assumed conditions to expectations by following paths representing the equivalence of logical transformations, the plausible sufficiency of heuristic transformations, or the antecedent-consequent relations of instrumental acts. At each point, a transformation links premises to expectations, and these expectations may become part of the premises for a later inference. In short, a proof maps a general model of cause-effect relations into a specific derivation of the expected consequences of the planned actions.

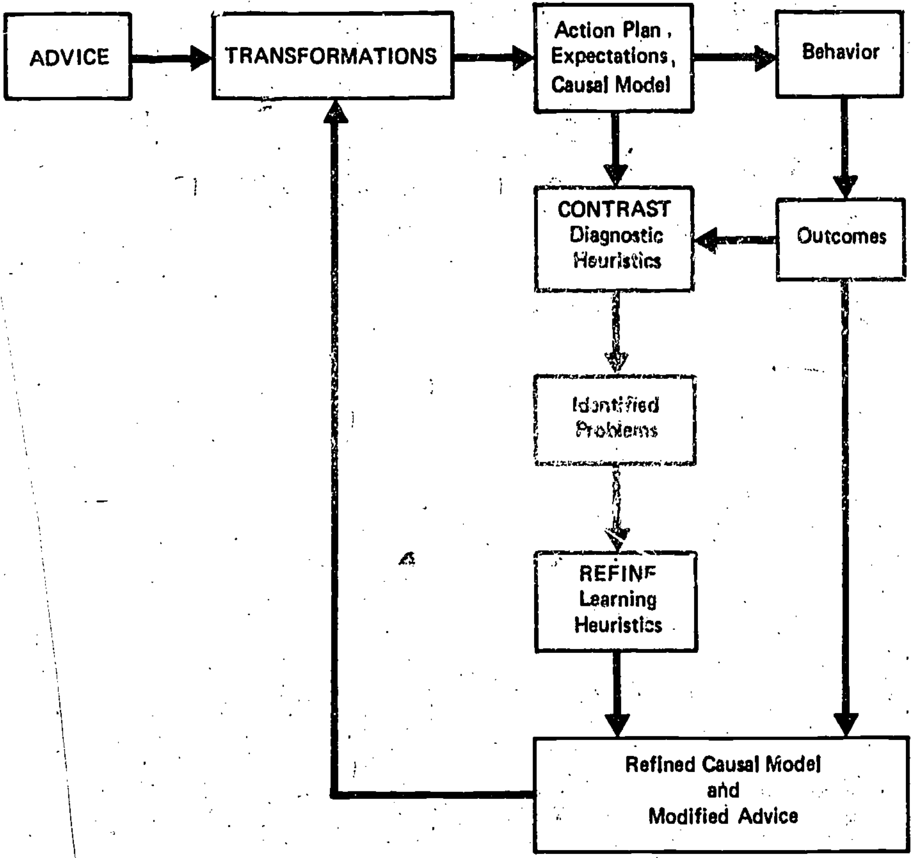
ADVICE
TRANSFORMATIONS
Action Plan, Expectations, Causal Model
Behavior
CONTRAST Diagnostic Heuristics
Outcomes
Identified Problems
REFINE Learning Heuristics
Refined Causal Model and Modified Advice

Fig. 4—Knowledge-refinement strategy

Diagnostic rules examine the proof in light of the evidence and identify hypothetical deficiencies in the knowledge base. A typical diagnostic rule is as follows:

Invalid Premise Diagnostic Rule. If an expectation is violated, find a premise in the proof of the expectation that is falsified by the data. If the false premise follows from some inference rule whose own antecedent premises (necessary conditions) are true, declare that rule faulty.

Learning rules, on the other hand, specify ways to modify heuristics to correct deficiencies. We have generated a large set of such rules to date. Two examples of learning rules are given below:

Require Implicitly Assumed Premise Explicitly. If an implicit assumption of a rule is falsified during proof analysis, add the premise to the required conditions of the rule and delete any other premises that it implies.

Guarantee Assumed Conditions. If an assumed premise is falsified during proof analysis, identify sufficient conditions for its validity and make these required conditions for the associated plan component.

Figure 5 demonstrates how these diagnostic and learning rules are used to refine the original "flush the queen of spades" plan as discussed above. Figure 5 also exemplifies the knowledge-refinement approach outlined in Fig. 4.

We have thus found many ways to evaluate a plan against observable outcomes to identify weaknesses, conjecture refinements, and evaluate these refinements experimentally. Very little of this work has been implemented, because of the vast number of possible learning strategies (see Table 5) and the wide variety of specific possible applications. Any efforts to implement these concepts in a realistically complex task will encounter considerable combinatorial difficulties. Each error may suggest several hypothetical bugs and fixes. Each of these will require independent empirical (or formal) validation, usually accomplished best by experimental testing. The need for testing hypothetical concepts and rules will lead to alternative knowledge bases and associated operational programs. Multiple systems of this sort are, of course, difficult to manage even in limited software-development environments.

SUMMARY

Once a plan is executed, much can be learned from a retrospective analysis. When advice is provided initially, two important things are missing that later support evaluation and discovery. The first new source of information is the actual situation description. The details of the actual situation in which the plan executes reveal and implicitly define important special cases that the general operationalization overlooks. Second, having acted, we can see the true effects of our behavior on the environment. This provides sources of confirmation or disconfirmation of parts of our plans, which then stimulate focused efforts at diagnosis and knowledge refinement. These provide numerous opportunities for concept formulation, and each, in turn, initiates a new cycle of knowledge acquisition, knowledge programming, and knowledge refinement.

PLAN:
Flush queen of spades:

If player P takes the lead
and P doesn't have the queen of spades
then P continues leading spades

Expectation:
P doesn't take the queen of spades.

Proof of expectation:

1. Player P takes the lead	Premise (condition of plan)
2. P doesn't have the queen of spades.	Premise (condition of plan)
3. P continues leading spades.	Premise (action of plan)
4. If player P takes the lead and P doesn't have the queen of spades and P continues leading spades then opponent will play queen of spades.	Heuristic rule
5. Opponent will play queen of spades.	Derived premise from 1, 2, 3, 4
6. If an opponent plays queen of spades then the opponent wins the trick and opponent takes the queen of spades.	Heuristic rule
7. Opponent takes the queen of spades	Derived premise from 5, 6
8. If opponent takes the queen of spades then player P doesn't take queen of spades.	Heuristic rule
9. Player P doesn't take queen of spades.	Derived premise from 7, 8

Behavior in actual play: P leads king of spades;
Opponent plays queen of spades.

Outcome: P wins the trick; P takes the queen of spades.

Expectation of "Flush queen of spades" plan is violated.

Apply diagnostic rules to identify problems:
Using "Invalid Premise" diagnostic rule, the derived premise in Statement 7 is falsified by the data. The inference rule used to derive this false premise is the rule specified in Statement 6. Its premise is true, but its conclusion is false. Declare this rule faulty.

Apply Learning Rules to modify plans and heuristics:
Using "Guarantee Assumed Conditions" learning rule, the system looks for other rules in the knowledge base that identify conditions for inferring Statement 7. It may find, for example, the inference rule:

If opponent player plays a high card C
and player P plays below C
then opponent wins trick and takes C

In our current example, C is the queen of spades.
This rule now replaces the faulty rule in Statement 6 with the new premise

Player P plays below the queen of spades

added as a premise to the plan and the proof. The resultant plan is

If player P takes the lead
and P doesn't have the queen of spades
then P continues leading spades
below the queen of spades

Fig. 5—Knowledge-refinement example

Table 5

KNOWLEDGE-REFINEMENT STRATEGIES

Type of Problem	Refinement Strategy
1. Excess generality	Specialize the rules, using case analysis, proof analysis, concept hierarchy specializations.
2. Excess specificity	Generalize the rules, using equivalence of cases, proof analysis, concept hierarchy generalizations.
3. Concept poverty	Create new concepts, by characterizing a particular problem, adding its definition, consequences, and proposed solution to knowledge base (e.g., "sacrifice," "safe" distribution).
4. Invalid knowledge	Correct faulty advice, using proof analysis, diagnosis, and refinement.
5. Ambiguous knowledge	Explore alternative interpretations and prune those that produce least desirable effects.
6. Invalid reasoning	Correct faulty operationalizations, using proof analysis, diagnosis, and refinement.
7. Inadequate integration	Develop comprehensive operationalizations that satisfy multiple pieces of advice simultaneously; sequentially order separable criteria to satisfy most important considerations first.
8. Limited horizon	Elaborate plans to incorporate contingencies and predict, monitor, and remember their outcomes; wherever possible, prefer dynamic operationalizations to static ones.
9. Egocentricity	During planning, consider what others are likely to do; use your own plans to model what you would do in their places; then monitor their behavior to assess its consistency with your model.

IV. CONCLUSIONS AND RECOMMENDATIONS

FINDINGS

In today's environment, major advances in AI arise primarily in conjunction with knowledge-engineering research. In this area, the power of intelligent systems derives primarily from the knowledge of human experts. The primary bottlenecks in the construction of intelligent systems are formulating knowledge for programmers, converting the knowledge into effective procedures, and iteratively evaluating a program's behavior, modifying the knowledge, and reimplementing the corresponding program code.

We have formulated a framework for exploring solutions to these problems which provides a basis for experts to express domain knowledge in terms of natural domain-specific concepts. This requires a formal knowledge-representation scheme and a substantial set of built-in primitive concepts from which the specific domain concepts are constructed. Once the concepts are defined, the expert can express two kinds of advice about the behavior of the program. Constraints specify restrictions on allowable behavior, while heuristics prescribe desirable modes of behavior. These may be ambiguous, incomplete, or even inconsistent.

This advice is converted into a working program through a process of operationalization, which transforms constraints and heuristics into effective procedures. In this process, the current program uses the expert's supplied knowledge along with about 300 transformation rules. Some of these reformulate expressions in equivalent terms, for example, by substituting a definition for some specific term. Some of the rules prescribe sufficient or approximately sufficient means of achieving ends. Finally, the operationalization process uses instrumental reasoning to predict effects of potential actions or to reason backwards from desired effects to sufficient conditions and actions.

In the process of operationalizing advice, a plan is developed that prescribes a sequence of actions required to accomplish the goals and satisfy the constraints. To develop this plan, the knowledge programmer employs a causal model to establish a "proof" of the plan's expected effects. When the plan is executed, new data about the situation and the effects are obtained. By contrasting observations with expectations and premises in the proof, diagnosis rules indicate faulty components of the plan. These in turn lead to plausible refinements to the plan and corresponding changes to the knowledge base. These refinements, in turn, reinitiate the cycle of operationalization, execution, and evaluation.

We have found this paradigm quite valuable as a source of new ideas and methods for knowledge acquisition and refinement. We have implemented only the operationalization component and have experimented with several knowledge representations in different tasks to develop diagnostic and learning rules. We have not yet converged on a small set of rules for any aspect of this paradigm. We have approximately 300 rules of operationalization for two tasks (hearts and a simple music composition task), and fewer than 100 diagnostic and learning rules. However, we foresee these numbers increasing to as many as a few thousand. For example, many of Lenat's proposed general concept-discovery methods (e.g., generalization and specialization heuristics) seem to apply to behavioral tasks as well as to mathematics (Lenat, 1976; Lenat, 1977b). Our current ceilings have been imposed by funding and personnel limitations: We have found many more interesting and productive lines of investigation than we have had resources to pursue.

RECOMMENDATIONS

We recommend that the proposed research paradigm be adopted widely. It focuses on a set of learning problems that are considerably different from those addressed in most previous learning research in AI and cognitive psychology. Much previous research (our own included) addressed isolated concept, pattern, and rule-learning problems—tasks that seem fundamentally tied to limited applications. Although the number of potential applications for pattern- or rule-induction systems is large, most learning problems will arise in the context of more fully integrated intelligent systems. In these systems, capabilities will be required for recognizing patterns, gathering information, assessing uncertainties, trading off between multiple goals, satisfying a variety of constraints, and dynamically applying general principles to specific situations. These capabilities in turn create demands for both rapid knowledge programming and rapid refinements.

We also recommend that learning issues be approached within the broader context of purposive behavior. In this context, the value of knowledge derives from its capacity to contribute to goal attainment. Goal-oriented planning provides a basis for contrasting the expected effects of knowledge with actual effects. This in turn dictates what new knowledge must be produced and how to integrate it into a preexisting knowledge base. This type of teleological orientation strongly motivates and guides knowledge acquisition and refinement.[1]

Finally, we suggest an increased emphasis on the core research problems standing between our current state of technology and the capability of automatic knowledge programming and refinement discussed in this report. The primary research problems include (1) representations for concepts, constraints, and heuristics amenable to machine interpretation and semantic analysis; (2) translators for mapping natural domain descriptions into these knowledge representations; (3) operationalization and planning; (4) plan evaluation and proof analysis; and (5) knowledge-refinement and concept-discovery heuristics.

[1] A corollary to this recommendation argues that when constructing or modifying AI programs, we should try to analyze the reasoning involved. One step in this direction is to identify operators for transforming specifications into working code. (See Balzer et al., 1977; Barstow, 1977; Mostow & Hayes-Roth, 1979a.)

BIBLIOGRAPHY

Allen, J., *Anatomy of LISP*, McGraw-Hill Book Company, New York, 1978.

Anderson, R. H., "The Use of Production Systems in RITA to Construct Personal Computer 'Agents'," *SIGART Newsletter*, Vol. 63, 1977, pp. 23-28.

Anderson, R. H., and J. J. Gillogly, *Rand Intelligent Terminal Agent (RITA): Design Philosophy*, The Rand Corporation, R-1809-ARPA, February 1976.

Balzer, R., "A Mathematical Model for Performing a Complex Task in a Card Game," *Behavioral Sciences*, Vol. 2, No. 3, May 1966, pp. 219-236.

Balzer, R., N. Goldman, and D. Wile, "Informality in Program Specifications," *Proc. 5th Int. Joint Conf. Artificial Intelligence*, Cambridge, Massachusetts, 1977, pp. 389-397.

Barr, A., J. Bennett, and W. Clancey, *Transfer of Expertise: A Theme for AI Research*, Technical Report HPP-79-11, Stanford University, March 1979.

Barstow, D., "A Knowledge-Based System for Automatic Program Construction," *Proc. 5th Int. Joint Conf. Artificial Intelligence*, Cambridge, Massachusetts, 1977, pp. 382-388.

Biermann, A. W., and R. Krishnaswamy, "Constructing Programs from Example," *IEEE Trans. Software Engineering*, Vol. SE-2, No. 3, 1976.

Brown, J. S., and R. R. Burton, "Multiple Representations of Knowledge for Tutorial Reasoning," in D. Bobrow and A. Collins (eds.), *Representation and Meaning*, Academic Press, New York, 1975, pp. 311-349.

Buchanan, B. G., and T. Mitchell, "Model-Directed Learning of Production Rules," in D. A. Waterman and F. Hayes-Roth (eds.), *Pattern-Directed Inference Systems*, Academic Press, New York, 1978, pp. 297-312.

Buchanan, B. G., G. Sutherland, and E. A. Feigenbaum, "Heuristic Dendral: A Program for Generating Explanatory Hypotheses in Organic Chemistry," in B. Meltzer and D. Michie (eds.), *Machine Intelligence 4*, American Elsevier, New York, 1969, pp. 209-254.

Church, A., *The Calculus of Lambda-Conversion*, Princeton University Press, Princeton, 1941.

Davis R., "Interactive Transfer of Expertise: Acquisition of New Inference Rules," *Artificial Intelligence*, Vol. 12, No. 2, August 1979, pp. 121-158.

Davis, R., "Knowledge Acquisition in Rule-Based Systems—Knowledge About Representations as a Basis for System Construction and Maintenance," in D. A. Waterman and F. Hayes-Roth (eds.), *Pattern-Directed Inference Systems*, Academic Press, New York, 1978, pp. 99-134.

Davis, R., B. Buchanan, and E. H. Shortliffe, "Production Rules as a Representation for a Knowledge-Based Consultation System," *Artificial Intelligence*, Vol. 8, 1977, pp. 15-45.

Davis, R., and J. King, "An Overview of Production Systems," in E. W. Elcock and D. Michie (eds.), *Machine Intelligence 8*, John Wiley & Sons, New York, 1976, pp. 300-332.

Duda, R. O., P. E. Hart, N. J. Nilsson, and G. L. Sutherland, "Semantic Network Representations in Rule-Based Inference Systems," in D. A. Waterman and F. Hayes-Roth (eds.), *Pattern-Directed Inference Systems*, Academic Press, New York, 1978, pp. 203-221.

Fagan, L., "Ventilator Manager: A Program to Provide On-Line Consultative Advice in the Intensive Care Unit," Technical Memo HPP-78-16, Computer Science Department, Stanford University, September 1978.

Faught, W., D. A. Waterman, P. Klahr, S. J. Rosenschein, D. Gorlin, and S. J. Tepper, *EP-2: An Exemplary Programming System*, The Rand Corporation, R-2411-ARPA, February 1980.

Feigenbaum, E. A., "The Art of Artificial Intelligence: Themes and Case Studies of Knowledge Engineering," *Proc. 5th Int. Joint Conf. Artificial Intelligence*, Cambridge, Massachusetts, 1977, pp. 1014-1029.

Feigenbaum, E. A., B. G. Buchanan, and J. Lederberg, "On Generality and Problem Solving: A Case Study Using the Dendral Program," in B. Meltzer and D. Michie (eds.), *Machine Intelligence* 6, American Elsevier, New York, 1971, pp. 165-190.

Hayes-Roth, F., "Schematic Classification Problems and Their Solution," *Pattern Recognition*, Vol. 6, No. 2, Winter 1974, pp. 105-113.

Hayes-Roth, F., "Patterns of Induction and Associated Knowledge Acquisition Algorithms," in C. H. Chen (ed.), *Pattern Recognition and Artificial Intelligence*, Academic Press, New York, 1976a.

Hayes-Roth, F., "Representation of Structured Events and Efficient Procedures for Their Recognition," *Pattern Recognition*, Vol. 8, No. 3, July 1976b, pp. 141-150.

Hayes-Roth, F., "Uniform Representations of Structured Patterns and an Algorithm for the Induction of Contingency-Response Rules," *Information and Control*, Vol. 33, February 1977, pp. 87-116.

Hayes-Roth, F., "The Role of Partial and Best Matches in Knowledge Systems," in D. A. Waterman and F. Hayes-Roth (eds.), *Pattern-Directed Inference Systems*, Academic Press, New York, 1978a, pp. 557-574.

Hayes-Roth, F., "Learning By Example," in A. M. Lesgold et al. (eds.), *Cognitive Psychology and Instruction*, Plenum, New York, 1978b.

Hayes-Roth, F., and J. Burge, "Characterizing Syllables as Sequences of Machine-Generated Labelled Segments of Connected Speech: A Study in Symbolic Pattern Learning Using a Conjunctive Feature Learning and Classification System," *Proc. 3rd Int. Joint Conf. Pattern Recognition*, Coronado, California, 1976, pp. 431-435.

Hayes-Roth, F., P. Klahr, J. Burge, and D. J. Mostow, *Machine Methods for Acquiring, Learning, and Applying Knowledge*, The Rand Corporation, P-6241, October 1978.

Hayes-Roth, F., and J. McDermott, "Learning Structured Patterns from Examples," *Proc. 3rd Int. Joint Conf. Pattern Recognition*, Coronado, California, 1976, pp. 419-423.

Hayes-Roth, F., and J. McDermott, "An Interference Matching Technique for Inducing Abstractions," *Comm. ACM*, Vol. 21, No. 6, June 1978, pp. 401-410.

Heidorn, G. E., "Automatic Programming Through Natural Language Dialog: A Survey," *IBM J. Research and Devlopment*, Vol. 20, May 1976, pp. 302-313.

Heidorn, G. E., "English as a Very High Level Language for Simulation Programming," *Proc. ACM SIGPLAN Symposium on Very High Level Languages*, Santa Monica, California, 1974, pp. 91-100.

Lakatos, I., *Proofs and Refutations*, Cambridge University Press, Cambridge, 1976.

Lenat, D., "AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search," SAIL AIM-286, Stanford Artificial Intelligence Laboratory, Stanford, California, 1976. Jointly issued as Computer Science Department Report No. STAN-CS-76-570.

Lenat, D., "Automated Theory Formation in Mathematics," *Proc. 5th Int. Joint Conf. Artificial Intelligence*, Cambridge, Massachusetts, 1977a, pp. 833-842.

Lenat, D., "The Ubiquity of Discovery: 1977 Computers and Thought Lecture," *Proc. 5th Int. Joint Conf. Artificial Intelligence*, Cambridge, Massachusetts, 1977b, pp. 1093-1105.

Lenat, D. B., and G. Harris, "Designing a Rule System that Searches for Scientific Discoveries," in D. A. Waterman and F. Hayes-Roth (eds.), *Pattern-Directed Inference Systems*, Academic Press, New York, 1978, pp. 25-51.

Lenat, D. B., F. Hayes-Roth, and P. Klahr, *Cognitive Economy*, The Rand Corporation, N-1185-NSF, June 1979a.

Lenat, D. B., F. Hayes-Roth, and P. Klahr, "Cognitive Economy in AI Systems," *Proc. 6th Int. Joint Conf. Artificial Intelligence*, Tokyo, 1979b, pp. 531-536.

Miller, M. L., and I. P. Goldstein, "SPADE: A Grammar Based Editor for Planning and Debugging Programs," AI Memo 386, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, December 1976.

Minsky, M. L., and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, 1969.

Mitchell, T. M., "Version Spaces: A Candidate Elimination Approach to Rule Learning," *Proc. 5th Int. Joint Conf. Artificial Intelligence*, Cambridge, Massachusetts, 1977, pp. 305-310.

Mostow, D. J., and F. Hayes-Roth, *Machine-Aided Heuristic Programming: A Paradigm for Knowledge Engineering*, The Rand Corporation, N-1007-NSF, February 1979a.

Mostow, D. J., and F. Hayes-Roth, "Operationalizing Heuristics: Some AI Methods for Assisting AI Programming," *Proc. 6th Int. Joint Conf. Artificial Intelligence*, Tokyo, 1979b, pp. 601-609.

Newell, A., "Heuristic Programming: Ill-Structured Problems," in J. Aronofsky (ed.), *Progress in Operations Research*, John Wiley & Sons, New York, 1969, pp. 363-414.

Newell, A., and H. A. Simon, *Human Problem Solving*, Prentice-Hall, Englewood Cliffs, New Jersey, 1972.

Pople, H. E., "The Formation of Composite Hypotheses in Diagnostic Problem Solving: An Exercise in Hypothetical Reasoning," *Proc. 5th Int. Joint Conf. Artificial Intelligence*, Cambridge, Massachusetts, 1977, pp. 1030-1037.

Pople, H. E., J. D. Myers, and R. A. Miller, "The DIALOG Model of Diagnostic Logic and its Use in Internal Medicine," *Proc. 4th Int. Joint Conf. Artificial Intelligence*, Tbilis, USSR, 1975, pp. 848-855.

Samuel, A. L., "Some Studies of Machine Learning Using the Game of Checkers," in E. A. Feigenbaum and J. Feldman (eds.), *Computers and Thought*, McGraw-Hill Book Company, New York, 1963, pp. 71-105.

Shortliffe, E. H., *Computer-Based Medical Consultations: MYCIN*, American Elsevier, New York, 1976.

Soloway, E. M., and E. M. Riseman, "Knowledge-Directed Learning," *Proc. Workshop Pattern-Directed Inference Systems, SIGART Newsletter*, Vol. 63, 1977, pp. 49-55.

Stefik, M., "An Examination of a Frame-Structured Representation System," *Proc. 6th Int. Joint Conf. Artificial Intelligence*, Tokyo, 1979, pp. 845-852.

Sussman, G. J., *A Computational Model of Skill Acquisition*, American Elsevier, New York, 1975.

Sussman, G. J., and R. Stallman, "Heuristic Techniques in Computer Aided Circuit Analysis," Memo 328, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Cambridge, Massachusetts, 1975.

Vere, S. A., "Inductive Learning of Relational Productions," in D. A. Waterman and F. Hayes-Roth (eds.), *Pattern-Directed Inference Systems*, Academic Press, New York 1978a, pp. 281-295.

Vere, S. A., "Multilevel Counterfactuals for Generalizations of Relational Concepts and Productions," Technical Report, University of Illinois, Chicago Circle, 1978b.

Waterman, D. A., *Rule-Directed Interactive Transaction Agents: An Approach to Knowledge Acquisition*, The Rand Corporation, R-2171-ARPA, February 1978a.

Waterman, D. A., "Exemplary Programming in RITA," in D. A. Waterman and F. Hayes-Roth (eds.), *Pattern-Directed Inference Systems*, Academic Press, New York, 1978b, pp. 261-279.

Waterman, D. A., and F. Hayes-Roth (eds.), *Pattern-Directed Inference Systems*, Academic Press, New York, 1978.

Waterman, D. A., R. H. Anderson, F. Hayes-Roth, P. Klahr, G. Martins, and S. J. Rosenschein, *Design of a Rule-Oriented System for Implementing Expertise*, The Rand Corporation, N-1158-1-ARPA, May 1979.

Winston, P. H., "Learning Structural Descriptions from Examples," in P. H. Winston (ed.), *The Psychology of Computer Vision*, McGraw-Hill Book Company, New York, 1975.