# Toward AI Research Methodology: Three Case Studies in Evaluation

**COINS Technical Report 88-31**

Paul R. Cohen
Adele E. Howe

Experimental Knowledge Systems Laboratory
Department of Computer and Information Science
University of Massachusetts, Amherst, MA

**Abstract**

We describe the roles of evaluation in empirical AI research—in an idealized cyclic model—and in the context of three case studies. The case studies illustrate pitfalls in evaluation and the contributions of evaluation at all stages of the research cycle. We contrast evaluation methods with those of the behavioral sciences and conclude that AI must define and refine its own methods. To this end we describe several experiment "schemas" and many specific evaluation criteria; and we offer recommendations that we hope will encourage the development and practice of evaluation methods in AI.

This research is funded by the Office of Naval Research, under a University Research Initiative Grant, Contract N00014-86-K-0764 and ONR Contract AFOSR 4331690-01.

We are indebted to Scott Anderson, Carole Beal, Ross Beveridge, Carol Broverman, David Day, Mike Greenberg, Tom Gruber, Cynthia Loiselle, and Paul Utgoff for their comments on earlier drafts of this paper and to Peter Patel-Schneider for sending us guidelines for reviewers.

## 1 Introduction

Evaluation means making observations of all aspects of one's research, and perhaps making some judgments of merit based on those observations, and reporting both to the research community. In AI, evaluation should not be limited to observing and judging only how our systems perform. It is a vital part of all the stages of research that lead up to performance evaluation and that follow it. For example, the information retrieval system we will discuss in Section 3.2 performed well in comparison with traditional statistical systems; but after analyzing its behavior we could see plenty of room for improvement—many simple modifications that we expected would improve performance. This observation is not a performance measure, but tells us informally about the current state of a program, whether its performance is likely to improve or has reached a limit, how easy it is to modify, and so on. These observations are in some respects more important than simple performance measures because they tell us how research should proceed. Evaluations should provide ongoing guidance at all stages of the research cycle.

So ideally, evaluation should be a mechanism by which AI progresses both within and across individual research projects. It should be something we do as individuals to help our own research and, more importantly, on behalf of the field. We must observe and report our research carefully because our colleagues cannot. In other empirical fields, evaluation includes describing experiment designs, results and analyses. But this is difficult in AI because experiments often involve uncontrolled interactions of knowledge representations, inference methods, algorithms, and the user; and researchers typically do not have access to run-time data and programs, at various stages of development, from other laboratories.

Consequently, individual AI researchers have an unprecedented responsibility to observe, assess, evaluate, and communicate their results. Many do. But if a researcher doesn't tell us, for example, that a program was tuned to perform well on a particular data set, then we will never know. Tuning programs is not necessarily bad; in fact, it is a good empirical way to discover the best possible performance of a program (see Sec. 4.1). We are being pragmatic, not moralistic, when we use the terms "good" and "bad": It doesn't help AI if, for sound experimental reasons, one tunes a system, but then allows the research community to believe it will perform in general as well as it performs in the best cases. The community also needs to know the number of cases on which one's system has run, how much help it got from the user, the size of the system, how difficult it was to scale up, and so on. These general assessments, as well as the specific ones discussed in Section 2 and the Appendix, are owed by individual researchers to the field.

Evaluation is not standard practice in part because we don't have formal research methods, standard experiment designs, and analytic tools. Where will they come from? The fundamental challenge is that we must develop them ourselves. We must refine existing, rudimentary evaluation practices and develop new ones. But they must be appropriate to empirical AI; we are better off without them if they impede progress.[1] We advocate evaluation not out of envy for "real science," but because we believe AI needs evaluation to move forward. Thus, as a field, we are not obliged to adopt "scientific" evaluation criteria (see Sec. 4), but should design our own.

We approach the topic of evaluation from the perspective of AI researchers. Our evaluation criteria and methods are designed for empirical AI research (see [4,26,19] for complementary discussions). The purpose of empirical AI is to tell us about the behavior of AI systems—the interactions of knowledge representations, inference methods, algorithms and other components of systems that we could not anticipate from purely theoretical AI. We differentiate empirical AI from applied AI: Though both involve building systems, the goal of empirical AI is to develop and systematically experiment with new methods; in contrast, applied AI relies on methods we already understand and rarely contributes new ones (see [13,12,29] for discussions of evaluating applied systems).

This paper is offered as a first step in what we hope will become a discourse on evaluation in empirical AI. Section 2 describes a multistage model of empirical AI research and the roles of evaluation at each stage. Section 3 presents three case studies of evaluation. Section 4 discusses the relationship between evaluation and progress in empirical AI, contrasting it with other behavioral sciences. The Appendix presents five classes of evaluation criteria—a checklist for researchers.

[1]One of us was trained in cognitive psychology but quit because its reductionist, rigorous, one-hypothesis-at-a-time methodology seemed on balance to be counterproductive. Newell expresses this view in a paper called "You can't play 20 questions with nature and win" [25]; and Neisser relates a similar concern for the "ecological validity," or validity outside the laboratory environment, of psychology research in his book *Cognition and Reality*[23].

## 2 Evaluation of an empirical AI project

Empirical AI research can be viewed as a cyclic, multistage process.[2] The process is cyclic because analysis of our programs invariably suggests new problems (as illustrated by the arc from the last stage in Fig. 1 back to the first); and because evaluation at every stage can cause the researcher to reformulate or refine results from previous stages. For example, when designing a method for solving a problem (stage 1, Fig. 1), we often find that the problem is ambiguous, overambitious, or underspecified and must be refined (stage 1, Fig. 1).

The model of empirical AI research in Figure 1 is idealized because not all research includes all these stages, and, more importantly, researchers don't evaluate their work at each stage. In this section we will discuss the advantages of evaluation at all stages of one's research, that is, we will show why it is worth following this idealized model. Toward this goal, we also suggest specific evaluation criteria for each stage of the model in Tables 1–5 of the Appendix.

**Stage 1: Refining a topic to a task** Empirical AI begins when researchers find particular topics fascinating. The first stage of the research cycle involves simultaneously refining the research topic to a *task* and identifying a *view*. A task is something we will want a computer to do, and a view is a "pre-design," a rough idea about how to do it. This stage takes a lot of effort; researchers don't simply say, "Ok, we are fascinated by discovery, so let's try mathematical discovery as a task and heuristic search as a view" [20]. The process is iterative and directed by evaluations (as are all other stages in Fig. 1): Is the task significant, tractable, and representative of the phenomena we want to study? Is the view completely novel or adapted from a different task? Is it appropriate to this task? Is our goal to explore the efficacy of an extant view for a new task, or to explore a new view of an extant task? We call the latter "reformulation"—looking at a well-known task in a new way. Reformulations can be major (e.g., view problem solving in terms of domain-specific knowledge instead of weak methods) or more modest (e.g., view uncertainty as a problem to be solved instead of a phenomenon to be measured).

Evaluations during this stage direct one's own research, and also provide the AI community with carefully justified tasks, views, and reformulations. The evaluation criteria in Table 1 of the Appendix address two basic questions: can you justify the research task to yourself and to the community, and do you understand what will be

[2]Buchanan [4] describes a similar model.

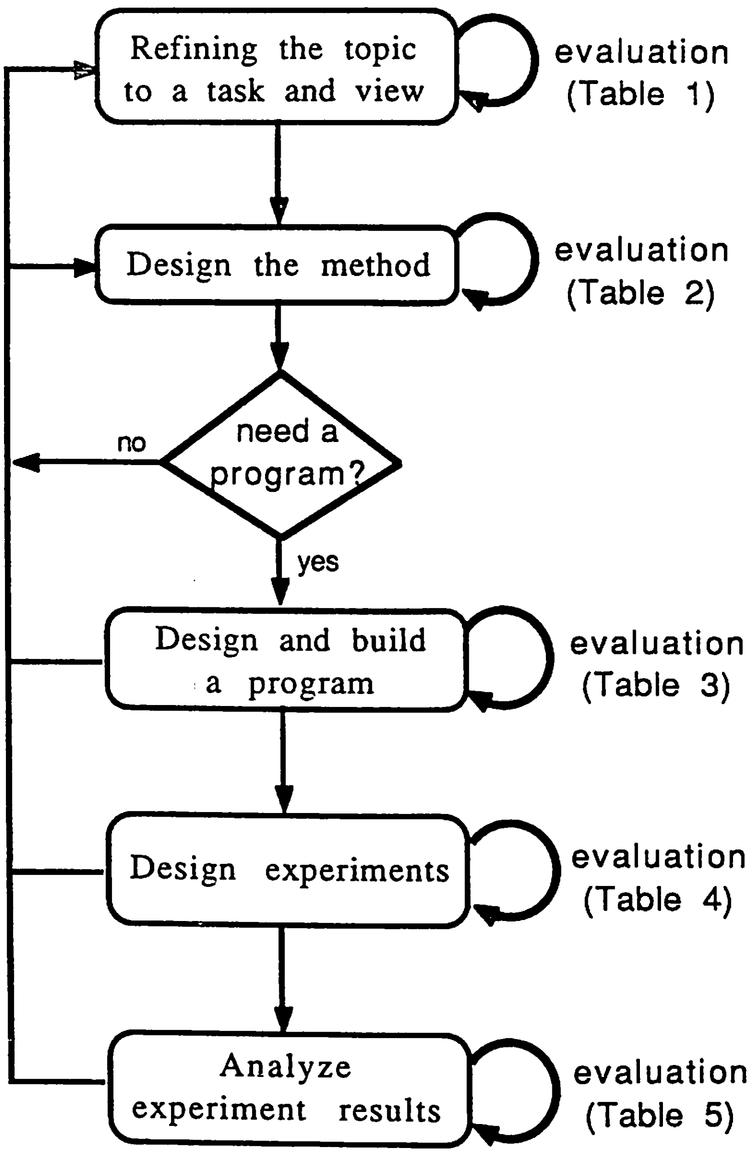

Figure 1: Cycle of Empirical AI Research

required to solve it?

**Stage 2: Design the method** At the next stage, one's view is refined to a method. The word "method" implies a single algorithm, such as $A^*$ [1] or candidate elimination [21], or Waltz filtering [31]. But frequently, the method for a task combines several algorithms and assorted knowledge structures. For example, the island driving method in Hearsay-II required many knowledge sources, data-driven and opportunistic control, and a novel communication structure [11]. Although this complexity strains the word method, we will maintain it to remind us that we don't jump immediately into building programs, but first decide *how* we want to solve tasks.

Designing a method is an iterative process guided by evaluation: what is the scope of the method, what are its underlying assumptions, does it rely on other methods, is it an improvement over existing technologies? How efficient is the method? How brittle is it? Are limitations inherent, or can the method be extended? Our field has few formal criteria for evaluating methods, yet the method is often the general contribution to the field. Table 2 in the Appendix presents criteria that assess how well you understand your method—its advantages and limitations.

In many cases, the method cannot be evaluated until it is implemented in a program. But let us first consider two cases in which programming is not necessary for evaluation. First, some methods can be evaluated analytically without programming (e.g., $A^*$ search, candidate elimination). The purpose of these evaluations is to tell the research community about the scope, efficiency, limitations, and other aspects of the methods. Second, a method may already be so well understood that neither implementing it nor evaluating it will tell us anything we do not already know. Now, in both cases, exploratory programming may help us refine the method we want to solve the task, but we distinguish this role of programming—refining a method—from programming for the purpose of experimenting with and evaluating a method. Unfortunately, exploratory programming drifts easily into building systems, and we begin to focus on solving the task, and forget that, from the standpoint of empirical AI research, the purpose of building systems is to tell us something about our methods that we don't already know and can't learn by analysis. We build too many systems and evaluate too few.

**Stage 3: Design and build a program** If the method requires programming not merely to implement it, but to understand whether and why it works, then we move on to the third stage in Figure 1. Here, the method becomes a design and then a program.

Although in practice this stage may be indistinguishable from exploratory programming in the previous stage, its purpose is different. Consequently, the evaluations we do at this stage are different. Evaluation, at this stage, mostly involves checking that your program implements as much of the method as you wished to test, and that its demonstration of interesting behavior is transparent (these two checks are operationalized in the criteria in Table 3 in the Appendix). Unlike the earlier evaluations of tasks, views, and methods, evaluation at this stage is primarily for the individual researcher, not for the community at large. Its purpose is to direct the implementation of the method in a program that can be evaluated.

**Stage 4: Design experiments** The fourth stage of the research cycle is to design experiments with the newly-implemented system. These experiments help assess the utility, generality and efficiency of the methods and their implementations. Criteria for evaluating experiments (as opposed to their outcomes) focus on whether test cases will be informative, that is, whether they span the range of abilities claimed for the method, whether sufficient numbers of test cases will be run, and whether the performance measures and standards are appropriate (the full set is presented in Table 4 in the Appendix). In practice, stages 3 and 4 are interleaved: one doesn't implement programs before thinking about experiments. Section 3.1 illustrates the pitfalls of designing a program without considering the experiments. The purpose of these evaluations is to convince the individual researcher and the research community that experiments are sound—that they demonstrate what they purport to.

**Stage 5: Analyze experiment results** The method has been implemented in a fully-instrumented program, the experiments are well designed, and now we can ask whether the system works and why it works. How does it compare to its performance standard? Did it perform differently than expected? Is it efficient? What are its performance limitations? What happens if we change the control strategy, or try a new set of test cases, or remove some of its knowledge? What if we manipulate several of these factors at once? After all, the point of building the program was to find out how the complex interactions of components of our method affect performance in many different conditions. Many research computer programs disappear shortly after they are written; their legacy is not their binary representation, but rather the new knowledge they reveal. The results of experiments and generalizations based on those results are the primary contribution of the research project to the community (the criteria for this stage are summarized in Table 5 in the Appendix). So the purposes of evaluation at this stage are to convince the research community of the viabilitity, performance and scope of one's methods, and to suggest further research.

In sum, evaluations at each stage in Figure 1 tell us whether to proceed to the next stage, repeat the current stage or return to a previous stage; and at most stages they also provide the research community with information about what we are doing and why. Evaluation informs our research by telling us, as soon as possible, that we have refined a fascinating topic to an intractable task, that our ingenious view of the task is too ambiguous to turn into a design, that our design doesn't address the most interesting aspects of the task, or that our implementation—though faithful to the design—simply doesn't work. If we complete one cycle of research, we need evaluation criteria to direct the next one. And if we are able, in the course of several cycles, to understand our task, then we need evaluation criteria to tell us how to refine the original topic to another, related one.

## 3 Case studies

This section illustrates the research cycle and the role of evaluation in the context of three case studies. Our first case study illustrates problems with evaluating knowledge-based systems, specifically a portfolio management expert system called FOLIO [8]. When we developed FOLIO, we didn't think through the details of the evaluations, so when we finished we discovered we couldn't do any convincing evaluations. The second case study focuses on the relationship between evaluation and the evolution of the GRANT system, specifically how our evaluations changed as we scaled up GRANT's knowledge base. Third, we examine the cyclic nature of the research model presented in Section 2. We describe how the results of analyzing Dominic, a mechanical engineering design system, led to more powerful versions of the system.

### 3.1 FOLIO

As an exercise in evaluation, FOLIO was only marginally successful. In prospect, we learned that unless one evaluates a research project at all stages, one may end up with a system that cannot be evaluated. Ironically, FOLIO seemed promising because we believed that portfolio performance was an objective evaluation criterion. What we failed to consider was that the *performance* of the portfolio was not what we should have been measuring—the appropriate criterion was whether the client's goals had been satisfied, and there was no objective measure of that criterion.

Investors often engage investment advisors or portfolio managers to help them manage their capital. FOLIO was an expert system that advised clients on *asset allocation* problems, in which clients' investable capital is divided among several *funds* [8]. For example, the assets of an elderly, retired individual might be allocated primarily to bonds and, in a lesser part, to blue-chip stocks. The proportion of the client's assets in each fund depends on many factors, some of which are provided by the client (e.g., risk tolerance, age, health, desired standard of living) and some of which are inferred (e.g., whether the client needs a hedge against inflation). To solve asset allocations problems, FOLIO divides clients' capital among several funds in such a way as to satisfy their stated and inferred goals and needs, and maximize after-tax income.

FOLIO's task was first to collect data from a client and infer how much the client cared about each of fourteen goals, then to use the desirability of these goals to determine the proportion of assets allocated to each of nine funds. We began to view the problem in two distinct ways: inferring the client's goals seemed like a standard classification task [5], for which we might use conventional expert systems methods; but constructing the portfolio to best achieve the client's goals seemed like an optimization problem. In fact, neither view is sufficient alone, so we adopted a hybrid view in which goals are inferred from client data, and then are passed to an optimization program that constructs the portfolio. We refined this view to a specific method: First, client data drive a rule-based expert system to infer goals and needs. These are represented on the right-hand sides of the rules as components of an objective function and linear constraints for a goal-programming algorithm [15]. Then the algorithm produces a portfolio. In short, the expert system configures an optimization program, which produces a portfolio.

We attempted to evaluate FOLIO's performance by what we call the $CC'$ script: one set of cases, $C$, is used to develop the program, typically to a high level of performance, and another set, $C'$, is used to test whether that level can be achieved in novel cases. The $CC'$ script requires a *measure* and a *standard*, both of which proved problematic in FOLIO. The obvious approach is to have the expert and FOLIO both produce portfolios for the N cases in $C'$, then generate three measures:

hit rate: the number of identical portfolios divided by N

miss rate: the number of portfolios generated by the expert but not by FOLIO, divided by N

false-positive rate: the number of portfolios generated by FOLIO but not by the expert, divided by N.

This approach, in which the expert's solutions are the standard of comparison, works well when problems have only one correct solution. But in asset allocation problems, many portfolios may satisfy the client's goals equally well. In fact, FOLIO and the expert never produced identical portfolios. In such cases, measures that depend on identity (e.g., hit rate) are inapplicable unless the expert can somehow be coerced into generating all acceptable solutions to each problem. Even this will not work unless there is some way to tell that all solutions have been generated (see Sec. 3.2 for a variant on $CC'$ that addresses this problem). So to evaluate FOLIO, we could really only ask the expert whether the system's portfolios were acceptable. This is a much weaker evaluation criterion than having the expert generate portfolios independently, because the expert may allow himself to be convinced that FOLIO's recommendations are acceptable (the variant on $CC'$ discussed in Sec. 3.2 also controls for this problem).

This raises the question of when expert judgments are appropriate standards of comparison for expert systems. Portfolio managers are notoriously inconsistent, raising the possibility that one's "expert" isn't really, and perhaps shouldn't be the standard of comparison.[3] One approach here is to have several standards of comparison. For example, the MYCIN system was evaluated against ten judges: five nationally-known experts, and five with varying levels of medical training [30]. The nonexpert judges were, in effect, a control condition to show that MYCIN is an *expert* system, that is, a system that solves problems that only experts can solve correctly. We tried this kind of group evaluation with other portfolio managers, but we could get no consensus about FOLIO's recommendations.

Before leaving FOLIO, let us consider what purposes its evaluation *should* have served. The research community doesn't care about the performance of yet another expert system: the community needs to know *why* a system works or doesn't, especially a system like FOLIO that merges AI and operations research (OR) techniques. Its evaluation should have pointed out the advantages, disadvantages, and impediments in developing hybrid AI/OR systems. These assessments are not performance evaluations so much as comments on the viability of a new technology. We made some observations of this kind, but never thought to report them, because at the time we thought evaluation was limited to performance. For example, FOLIO's goal-programming algorithm often produced portfolios that, though optimal, were judged "extreme" by the expert. Where the expert would have mixed half a dozen funds to achieve a "balanced" effect, FOLIO would recommend, say, 60% extremely conservative bonds and 40% extremely

[3]This question should not imply a lack of confidence in FOLIO's consulting expert, who has been highly regarded by his colleagues and clients for many years.

risky stocks. Extreme solutions are characteristic of many optimization techniques, and may present impediments to hybrid AI/OR systems. We also observed that FOLIO's rule-based component was very hard to debug, because it was designed to configure the objective function and linear constraints of the goal-programming algorithm, which was itself a black box. Thus, given a bad portfolio, it was virtually impossible to tell which rules ought to be changed. Quantitative performance evaluations are certainly important, but in FOLIO they were difficult to obtain and believe. We failed to realize that, ultimately, the more qualitative assessments are more informative and valuable to the research community.

## 3.2 GRANT

GRANT finds sources of research funding given research proposals [6]. The principle difference between GRANT and most other information retrieval (IR) programs is that its retrieval algorithms finds funding agencies based on *semantic* matches between research proposals and the agencies' research interests. For example, if a research proposal mentions **hemoglobin**, GRANT will find agencies that support research on **blood**, even if they don't specifically mention hemoglobin in their statements of interest. This semantic match is judged potentially productive because hemoglobin is a component of blood, and agencies that support research on substances or phenomena often support research on their components.

GRANT performs a common IR task—a researcher describes his or her interests and GRANT suggests potential sources of funding—but it is based on an unusual view. The view is that funding agencies will be indexed by nodes in a large semantic network, so that researchers don't have to use the exact words an agency uses in its statement of interest, but can use semantically-related words. The agency may say "blood" and the researcher "hemoglobin," but they will be matched up anyway. The problem with this view is that chains of semantic relations can be found between any pair of nodes in GRANT's network, so it is possible to link a proposal that mentions blood with an agency that mentions, say, air, because blood is part of the respiratory system, which processes air. The view we finally adopted in GRANT is called *constrained spreading activation*: a proposal activates nodes in a semantic network, and activation spreads through the network only on particular paths, until it activates nodes associated with agencies. The specific method uses an agenda of active nodes and rules to prune spreading activation. For example, one rule says that you cannot spread activation first over a **component-of** relation (e.g., from **hemoglobin** to **blood**), then over a **has-component** relation (e.g., from **blood** to **leucocyte**), because if a researcher wants to study one specific component of a substance (hemoglobin), he or she probably does not want to study some other component of that substance (leucocytes).

We have evaluated GRANT extensively at all stages of its development, focusing specifically on the constrained spreading activation method. Performance evaluations were based on a variant of the $CC'$ script (see Sec. 3.1). Instead of having the expert and GRANT solve a common set of problems, we had the expert judge the performance of a "dumb" version of GRANT, then used these judgments to generate performance measures for a "smart" version. The dumb version spreads activation to *all* nodes that can be reached by following up to four relations from each of the original proposal nodes. We call this the dumb set. The smart version constrains this activation, and so finds a subset of the dumb set, called the smart set. The expert judged whether each of the agencies in the dumb set were appropriate or inappropriate. Since the dumb set is a superset of the smart set, the expert's judgments were the basis of the following measures on the smart set:

recall: number of agencies judged good by the expert and GRANT, divided by the number of agencies judged good by the expert (also called hit rate).

fallout: number of agencies judged good by GRANT and bad by the expert, divided by the number of agencies judged good by GRANT (also called false-positive rate).

These measures were adopted from the IR literature [28]. When possible, comparison studies should use as measures the established performance norms of the domain.

The smart/dumb approach is a good control for a problem we mentioned earlier, that when experts judge the performance of an expert system they may feel biased to accept marginal answers. This is especially problematic when the expert can construct a plausible explanation on behalf of the system; for example, in both FOLIO and GRANT the expert could say "yes, I can see a reason for selecting this fund (or agency) and since the system has a reason, I won't criticize it." In fact, the "reason" was usually illusory, a reflection of the expert's own post-hoc explanation and justification of poor performance. In such cases, comparison studies of performance must have a control condition—a set of test items that are expected to be wrong. In GRANT the control condition was the dumb set minus the smart set.

In the early days of the GRANT project, we were very encouraged by high performance. In a network of about 2000 nodes and 50 agencies we had roughly an 80%

recall and a 32% fallout rate. This is much better performance than ordinary keyword search, which may have a fallout rate as high as 90%.[4]

More recently, the numbers are much less gratifying. When we increased the size of the network to 4500 nodes and 700 agencies, the performance dropped to 67% recall and 71% fallout. In [7] we reported on several experiments to discover why performance dropped. Perhaps the most surprising result was that just three rules for constraining spreading activation accounted for 85% of the agencies the expert said were good and 42% of the agencies he said were bad. Clearly, these rules need to be replaced by others that apply in fewer cases but with more discriminatory power. When we tried this, in a very brief experiment, we improved performance slightly.

The GRANT project shows that evaluation is more than just proving your system works. Otherwise, we could have quit GRANT after we built our first, small, high-performance system. Why not quit there? Because we need to know whether our methods are equally effective when our systems are scaled up, and whether limitations on their performance are fundamental.

The first reason is important because almost all empirical AI systems are small. We rarely acknowledge this, so we can be misled by apparently glorious, very small results. For example, working in our lab last year a decision analyst and a plant pathologist set out to compare decision analysis with a standard expert systems approach to diagnosing root diseases. This brief empirical study lasted just four days. The first three were spent interviewing the plant pathologist to structure the decision analysis and acquire the required probabilities and utilities. We capitalized on this problem-structuring phase on the fourth day and, in a couple of hours, built an expert system with slightly greater functionality than the decision-analytic system. Later, the decision analyst and the plant pathologist published their conclusion, that decision analysis is a viable approach to building expert systems [14]. They acknowledged that the study was too small to say anything conclusive. The dissenters in the lab argued that building decision analyses is an impossibly slow process because the time required to get probabilities from experts increases combinatorially. Who was right? We will never know, because the expert system that duplicated the decision analysis contained *only nine rules.* One cannot draw any conclusions about the relative merits of two technologies when the systems are so small.

[4]For example, the keyword system used by the Office of Research Affairs at the University of Massachusetts, for whom we built GRANT, finds roughly 200 agencies for each search, of which only 5 or 10 are worthwile. GRANT, in the early days, would find about 15 agencies of which at most 5 were *not* worthwhile.

Often, a technique that works on a small problem will not work on a larger one. AI has a responsibility to at least consider the question of whether techniques will scale up. It isn't necessary to build a system that is 500% bigger, as we did in GRANT, if one can address the question analytically. For example, we know from the mathematics of decision theory that the number of probabilities required from an expert will increase combinatorially unless the decision analyst structures the problem very carefully. Thus we can say analytically that scaling up depends on the skill of the decision analyst. In either case, empirically or analytically, we must address the question.

The second reason to continue a project after it has succeeded is to find and explain the limitations of a method. Had we quit the GRANT project when it had an 80% recall and a 32% fallout rate, we would never have known whether these rates are inherent, or could be improved, and if the latter, at what cost. Whenever one invents a new technique, such as constrained spreading activation, one must find its bounds. Where does it break, and why? It is not sufficient to demonstrate that it works—that is only half the story. Unfortunately, AI researchers rarely do the other half.

## 3.3 Dominic

Dominic is a long-term research project to investigate automated, domain-independent mechanical engineering design. It is based on the view of design as *iterative redesign* [10].[5] In iterative redesign, a rough initial design is gradually refined in a four-step cycle. The first step is to suggest relatively small design changes that are intended to improve one facet of performance; the second is to predict the effects of those changes on overall performance; the third is to modify or replace the proposed changes; and the fourth is to implement changes that are predicted to improve performance. This iterative cycle produces hill-climbing search tailored to problems in which the exact shape of the hill is unknown and the cost of taking a step may be high. The hill is described by design variables and design goals. At each step, the value of a design variable is changed to produce a favorable change in the performance on a particular design goal. The levels of performance on individual design goals are combined into an overall evaluation of the design, which is the height of the hill at that position.

Dominic's iterative redesign method divides the design process into a series of small decisions: which design goal to work on, which design variable to change for that goal,

[5]Jack Dixon, a professor of mechanical engineering at University of Massachusetts, is the principal investigator. We participated in the development of Dominic-I and Domineering. Jack Dixon and Mark Orelup have continued the project.

which new value to pick for the design variable, whether to modify, accept or reject the proposed change, and when to stop. To make these decisions, Dominic-I, the first program to result from the project, relied on two kinds of knowledge [16]. The first, called the *dependency order list* assigns precedence to design variables. The second, called the *dependency table*, relates changes in design goal values to changes in design variable values. Many cells in the dependency table are empty or approximate initially, but Dominic can update them as it runs.

Dominic-I performed adequately in two domains compared with the designs of students, experts, and problem-specific programs. It was always better than the students, usually better than the experts and sometimes better than the problem-specific program.

But Dominic-I's principle contribution was less its performance than its utility as an experimental environment for testing our ideas about *control* in iterative redesign. In Dominic we followed closely the research cycle presented in Section 2. Thus, the next phase of the project, given the working program, was to design, run, and analyze the effects of control on performance. This involved instrumenting the program, and adding explicit mechanisms to allow us to easily reconfigure the program with alternate control strategies. We ran experiments on 125 different configurations and analyzed their effects on Dominic-I's performance, which was evaluated on the quality of its designs, the time required to find the best design, the number of implemented design changes that decreased performance (instead of improving it as expected), and other measures related to output results and search efficiency. The analyses of these results suggested improvements in Dominic that led directly to two other programs in the Dominic family: Domineering and Dominic-II.

Domineering was built to learn the best configuration of Dominic-I for particular design domains [17]. In effect, Domineering is Dominic-I applied to itself. Dominic is used to design the best configuration of Dominic. The alternate control strategies mentioned above provided the design variables, and the performance criteria provided the goals. Domineering would configure Dominic-I, run it on a problem from the domain in question, observe its behavior on the performance criteria, and redesign Dominic-I's configuration based on learned relations in the dependency table. Domineering did produce configurations of Dominic-I that had better performance, but it required tremendous amounts of processing. This precluded analyses as detailed as those undertaken for Dominic-I. Even so, the program was clearly an evolutionary dead-end: it couldn't give us a better Dominic except by enormous effort. But because Domineering demonstrated the efficacy of configuring Dominic for particular problems, it suggested a less radical approach with similar benefits.

Evaluations of Dominic-I and Domineering convinced us that Dominic's configuration—its control strategies—strongly affected its performance. In terms of the research model discussed earlier (Fig. 1), we had completed the last stage and were poised to begin the cycle anew. For Orelup and Dixon, who continued the project, Dominic-II [27] became the focus of the new cycle through the model. Its task was more ambitious: it monitored its own performance and dynamically modified its control strategy to maintain high levels of performance. The view of design as iterative redesign was refined and elaborated; Orelup and Dixon identified six pathological behaviors that arose in Dominic-I, and six control strategies to apply individually and in sequence to fix the problems. To specify a method, these pathologies were operationally defined and two algorithms were developed: The first detected the pathologies in Dominic-II's design behavior, and the second determined which of the six strategies to apply to correct the problems. At this point, the Dominic-II project was at the end of the second stage of the new cycle. Clearly, it engendered a set of hypotheses about the efficacy of dynamic control that could not be tested except by implementing them in a program. Dominic-I was modified accordingly. Orelup and Dixon then tested the system on 27 cases in five domains (hydraulic cylinders, I-beams, post and beams, V-belts, and solar heating systems). All the cases were presented to both Dominic-I and Dominic-II. This comparative experiment design demonstrated that, in Dominic, dynamic control significantly improves performance: In all design domains, Dominic-II generated more designs, often of better quality and in fewer iterations, than Dominic-I.

The Dominic project illustrates the iterative nature of empirical AI research and the importance of evaluation. Evaluations tell us that Dominic-II was a significant improvement over Dominic-I; but more importantly, Dominic-II probably could not have been designed without the information provided by evaluations of Dominic-I and Domineering.

## 4 Discussion

We think of progress in AI, and thus the purpose of evaluation, in terms of *continuity, replication*, and *generalization.* Continuity within a laboratory, as we saw in the Dominic project, means that evaluations of each research project motivate the next. Continuity from one laboratory to another often begins by replicating results from the original laboratory (i.e., solving the same problem or one with similar characteristics). The pragmatic reason for this is probably that each empirical AI project has a large software base that, if not copied directly from another laboratory, must be reimplemented. Its fortuitous methodological consequence is that slightly different problems are solved in slightly different ways, making replication and generalization possible.

Consider this hypothetical case of replication and generalization:

> One laboratory builds a system to diagnose electrical faults, then another builds a similar system for diagnosing chest pain. Evaluations of the first system show that its control strategy, which depends on a causal model of the behavior of electrical circuits, blows up when faults can have many causes. The problem is addressed in the second system by providing probabilistic rankings of, in this case, causes of chest pain. Eventually, a visiting Bayesian formulates the control strategy in terms of a decision analysis.

Although we can all think of examples of this kind of progress in empirical AI, we shouldn't delude ourselves that it emerges from a rigorous methodology as it does in other sciences. Where other sciences have standard experimental methods and analytic techniques, we have faith—often groundless and misleading—that building programs will somehow be informative. Where other sciences expect specific aspects of research to be presented (e.g., hypotheses, related research, experimental methods, analyses and results), empirical AI has no comparable standards.

Empirical AI suffers by comparison with established sciences because its experimental methods are tacit; because its general cycle of research, described above, is misperceived as "just building programs" and thus is confused with applied AI; and because the methods and statistical analysis techniques that have become associated with "the scientific method" are largely inapplicable. Although empirical AI is a behavioral science that, like psychology, economics, and sociology, is concerned with thought and action in intelligent agents, the established experimental methods of these fields are inappropriate. Unless we understand why this is, and its implications for replication and generalization, we run the risk of self-defeating "science envy"—the sense that we are just a bunch of ingenious programmers, not *real* scientists—when we should be refining and inventing methods that are appropriate to empirical AI.

The following excerpt highlights differences between experimental methods in the established behavioral sciences and empirical AI. Fortuituously, the excerpted research draws on several fields, including physiological, abnormal, and developmental psychology.[6] Hereafter, we refer to the excerpt as the autism article.

[6]Our own experimental methods are, of course, impeccable: We obtained the first sentences of this selection by opening a random volume of the journal Cognitive Development to a random page.

> We wanted to determine whether a specific relationship exists between language ability and pattern of hemispheric specialization in autism.... Averaged cortical evoked responses to speech and nonspeech stimuli were recorded from the left and right hemispheres of autistic children and age-matched normal children. The evoked-response protocol was designed to be similar to that used by Molfese (1975) with normal infants. ... To assess whether the autistic and normal groups differed in their mean amplitudes of the N1 and P2 components, a multivariate approach to repeated-measures analysis of variance was used. ... The MANOVA ... yielded a significant overall effect, $F(4,29) = 3.57, p < .02$.[9]

Although it is difficult to tell exactly what is going on here (since we extracted very small parts of the article), one can see fragments of an established format for journal articles, established experimental methods (the reference to Molfese's procedure) and established statistical analysis techniques (the MANOVA). The experiment design incorporates the fundamental idea of a control condition (age-matched normal children). More fundamental still, and implicit in the statistical analysis, is the idea that hypotheses about causal relations (e.g., between language ability and pattern of hemispheric specialization in autism) are accepted if evidence for them could not have come about by chance; and that accepting them is actually an inductive generalization from a sample (e.g., autistic children) to the population from which the sample was drawn.

All this methodology is in service of a larger goal, in this case to find out about hemispheric specialization and the language abilities of autistic children. A closer look shows that the purpose of the research is just to demonstrate that a relationship *exists* between hemispheric specialization and language in autistic kids. Whereas this article—and much research in the behavioral sciences—is concerned with teasing apart the components of behavior and their causal interrelationships, empirical AI is concerned with putting all those components together in one box to produce behavior. This fundamental contrast is echoed in completely different styles of experimental research.

In the behavioral sciences, the basic question is "Why do organisms (or organizations) perform this way?" It is answered by two very general methods. One involves a broad search for factors that influence behavior, and is facilitated by statistical "discovery" techniques such as factor analysis, multidimensional scaling, and cluster analysis. The more common approach is called statistical hypothesis testing. The idea is to isolate a very small number of causal hypotheses in an experimental condition (typically less than three), and demonstrate performance differences between this condition and its corresponding control condition. Statistical hypothesis testing should not be confused with simply collecting descriptive statistics. It is actually a form of inductive inference. For example, the autism article set up a comparison between autistic and age-matched normal children (the experimental and control groups, respectively), measured differences in the mean amplitudes of N1 and P2 components (the dependent variable), and found by a MANOVA procedure a significant difference at the $p < .02$ level. "Significant" means that the difference was extremely unlikely—a probability of less than 2%—to have happened by chance. Since the result was obtained from a statistical and presumably representative sample, one can then inductively generalize the result to the population; in this case, to autistic children.

In empirical AI the basic question is "What knowledge, algorithms, representations, ..., and control strategies do we need to make an organism (or organization) perform this way?" The basic method, as we noted earlier, is an evolutionary cycle of tasks, views, and implementations. We demonstrate empirically that interactions of particular components will yield particular kinds of behavior. Our task is not to find out how the average human organism (or organization) works; but rather, to build artificial systems that work in particular ways. Because we are not trying to reduce complex phenomena to their causal antecedents, we do not need to run large groups of subjects in experimental and control conditions, testing hypotheses that differences between the conditions are due to chance.

This comparison is not intended to imply that all behavioral sciences besides empirical AI are entirely reductionistic. An obvious counterexample is the work of Jean Piaget, whose structuralist psychology (or, as he preferred, "genetic epistemology") has much in common with AI (e.g., [2] Ch. 7). Piaget's early work with his own children asks essentially the same question as we ask in empirical AI: "If I was designing an organism to behave this way, what internal structures would it need, and how would they develop?" Nor do we mean to imply that statistical discovery and hypothesis testing have *no* place in, say, Piaget's work or in empirical AI. Indeed, thousands of experiments have been run to find out how Piaget's "design for a child" performs in different conditions; and in empirical AI we would expect statistical hypothesis testing to help us tease apart the complex and unanticipated interactions of components of our systems. But we *are* saying that the experimental designs and analytic techniques that are associated with the behavioral sciences are fundamentally reductionistic, and so are not much use unless one's goal is to identify the components (and to a limited extent, their interactions) of complex behavior.

But since AI systems are unique artifacts, and we rarely run statistical experiments on them, how general can our empirical conclusions be? When one tests hypotheses by comparing, say, groups of undergraduates, one can be reasonably sure that the results will hold for the population at large. Can we ever be convinced that the results of an experiment on an individual AI system are general to other systems? The criterion for accepting a result in statistical hypothesis testing is that it almost certainly did not occur by chance. Do we have a criterion as convincing as this in AI?

To answer these questions, note that statistical experiments yield an *inductive* form of generality; an effect demonstrated in 100 undergraduates occurs in all undergraduates. Another kind of generality has been called *explanation-based* [22] or *deductive* [18]. We believe the sun will rise not because it always has, but because we can explain or deduce that it will from an underlying theory. The most convincing generalizations in AI are of this kind. For example, most AI researchers believe that data-interpretation tasks such as vision and speech understanding require large amounts of world knowledge. We believe this is a general result because it can be explained or deduced from an underlying theory; in this case, search. Unconstrained data-driven interpretation generates intractable search spaces, and any constraints on the process reflect world knowledge, therefore world knowledge is required for interpretation tasks. Inductive arguments are helpful, too. We have many examples of the importance of world knowledge in vision, speech understanding, human perception, psycholinguistics, and so on. But the point is that hypotheses in AI can be generalized deductively via underlying theories. We do not require inductive or statistical generalization.

In sum, the purpose of evaluation is to promote continuity, replication, and generalization in empirical AI. We discussed how evaluation drives the five stages of the basic research cycle, that is, how it produces continuity within a single research project. Unfortunately, we have only tacit, informal evaluation methods to promote continuity, replication, and generalization *across* research projects. It is essential to recognize and standardize these methods, because those of the established behavioral sciences are not appropriate.

## 4.1 Experiment Designs

In the previous sections we described the empirical AI research cycle and evaluation criteria—the components of a skeleton of a methodology. Although the last three stages of the cycle advocate the design and analysis of experiments, they don't tell us how to do them. AI is evolving stylized experiment "schemas" that, if they could be standardized, could guide researchers' experimental work and provide a shorthand for discussing results. We briefly describe five such schemas:

**Comparison studies.** The basic form of a comparison study is that we select one or more *measures* of a program's performance, then both the program and a *standard* solve a set of problems, and finally the solutions are compared on the measures. For example, we may compare the average number of subgoal violations generated by one planning program on a set of problems (the measure) with the same measure on another extant program (the standard). Typically, the programs will implement different methods, or they could be different configurations of a single program. The comparison of Dominic-I and Dominic-II (Sec. 3.3) illustrates this kind of study.

Variations on the basic form depend on what you want to demonstrate. For example, if you want to measure whether the program's performance is consensual, you may compare the program to a panel of human experts. You may also include novices—an interesting control condition to ensure that successful performance requires expertise.[7] Sometimes the performance of a program can be compared with objective, recognized standards. Normative theories, such as probability theory, provide one kind of standard; for example, some researchers argue that because human experts are incapable of integrating probabilistic information consistently, their performance should not set standards. Another kind of standard is provided by real or simulated worlds; we might evaluate a complex planner by seeing whether it generates plans that succeed in the world. All these examples suggest that our measures and standards depend heavily on what we want to demonstrate and, ultimately, on our research goals.

A related scheme, though not strictly a comparison study, has humans judging or scoring the program's performance. This happens when we need to measure whether programs get the "right" solution, but the test problems have so many acceptable solutions that a program and a standard cannot be expected to generate the same ones. FOLIO and GRANT (Sections 3.1 and 3.2, respectively) provide examples of this kind of study.

**Ablation and substitution studies.** We can evaluate the contribution of individual components to the performance of complex systems by removing or replacing those components. Removing components (called "ablation" [24]) is informative in systems that can solve problems without them. For example, one configuration

[7]Shortliffe ran a panel of experts and novices in his studies of MYCIN [30].

of Dominic-I had no dependency order list to tell it which design variables to change first. It takes little insight to predict That this will have *some* effect; the goal is to find out whether performance on all types of problems is equally affected by the presence of the dependency order list, and if not, what interactions between it and the problem type explain the variance.

Many AI systems are so brittle that they collapse when components are removed. In these cases we might substitute "dumb" components for those we hope to show are "smart"; for example, we might substitute an exhaustive control strategy for a sophisticated opportunistic one. Dominic-I required a dependency table to predict the effects of changing design variables, but in one experiment we substituted coarser values to assess the effects of their accuracy.

**Tuning studies.** By tuning a system to perform as well as possible on a set of test data, we can learn how much performance can be improved, how difficult it is to achieve, and whether the resulting system can still solve other test cases. From a research perspective, it seems wasteful and potentially misleading to tune systems just to increase their performance, without addressing these questions.

**Limitation studies.** By testing a program at its known limits, we can better understand its behavior in adverse conditions. We can push a program to its limits by providing imperfect data (rearranged, noisy, incomplete, or incorrect), restricted resources (computation time or available knowledge), and perverse test cases.

**Inductive studies.** One way to support claims of generality is to solve "new and different" problems. If we claim that Dominic is general, then we may want to run problems in many areas of mechanical design—pulley systems, I-beams, extrusions, and so on. Even if we don't claim a program is general, we must at least test it on problems other than those we used to develop it.

## 4.2 Recommendations

We begin with a now-familiar theme: AI researchers must evaluate their work more thoroughly and report both the results and how they were obtained. The latter will add to a common stock of evaluation techniques and will eventually flesh out the methodological skeleton.

At the same time, AI journals and conferences must welcome papers that discuss, in more detail and with more background than we can offer here, aspects of our evolving methodology. In particular, we hope to see more systematic, exhaustive analyses of schemas for comparison studies and the other schemas discussed above, and others we haven't yet considered. We also need further discussion of what it means for results to be general. This might build on the inductive/deductive distinction outlined earlier. But in any case, our field must consider how to justify the claim that projects in different labs, task domains, and programming environments demonstrate the same thing. We also need to see critical assessments of the methodological role of programs. Our impression is that researchers rarely glean enough from their programs to justify the effort of building them. Even if they do, the effort cannot be justified unless results are communicated to the research community. And the role of programs is just part of a broader debate on the empirical AI research cycle, which requires considerable elaboration beyond the sketch in Section 2. These are just a few of the methodological issues that ought to be discussed in print.

Having raised the issue of what gets published, let us consider some uncommon but important kinds of papers. AI systems take so long to build that we are surprised to see so few reports of experiments with extant systems. A good model of this kind of work is a book on experiments with MYCIN, edited by Buchanan and Shortliffe [3]. We also hope to see more papers on negative results—algorithms that don't work in particular cases, systems that perform *less* well as they become more knowledgeable, cases where scaling up causes problems. When, at a recent conference, we discussed the reasons that GRANT performed less well when it was scaled up, someone in the audience remarked how refreshing it was to see some "dirty laundry".[8] An odd phrase, dirty laundry, suggesting that there is something nasty about negative results. When did you last read an AI paper that said something didn't work?

Researchers will not document the limitations of their methods unless reviewers, program committees, and editors endorse papers on negative results, as we believe they must. These groups are also responsible for scrutinizing positive results. One recommendation is that if a paper doesn't answer a satisfactory number of the questions in Tables 1–5 in the Appendix, or comparable questions that are better suited to the subject of the paper, then it should be rejected. We would hope that program committees and editors would publish the criteria by which they evaluate papers, and that they would be more informative than the half-dozen buzzwords one often sees—original, thorough, thoughtful, well-written, and so on[9]. Another recommendation,

[8]The Third Annual IEEE Conference on Artificial Intelligence Applications. Orlando, FL. February, 1987

[9]Lately, the machine learning community has mentioned evaluation criteria explicitly in Calls for Papers, and Langley has advocated evaluation in his editorials [19].

more appropriate to journals than conferences, is that reviewers should provide detailed feedback on why papers are rejected or conditionally accepted, so that authors can run the recommended experiments and resubmit their papers. Thus, the primary responsibilities of reviewers and editors are to encourage discussions of evaluation criteria, publish them, insist the criteria are met, and provide guidance when they are not.

Our final recommendation is that we should keep the purposes of evaluation firmly in mind and not let it become an end in itself. In many disciplines you can't publish "just ideas." But even when your idea is distilled to an experimental hypothesis, and an enormous experiment is run, and profound results are obtained, your paper can still be rejected for petty methodological infractions. This isn't what we want for empirical AI. The purpose of evaluation is not to hold the field back, but to propel it forward.

## References

[1] Avron Barr and Edward A. Feigenbaum, editors. *The Handbook of Artificial Intelligence, Volume 1.* William Kaufmann, Inc., Los Altos, CA, 1981.

[2] Margaret A. Boden. *Jean Piaget.* Penguin Books, New York, 1979.

[3] B. G. Buchanan and E. H. Shortliffe. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley, Reading, MA, 1984.

[4] Bruce Buchanan. *Artificial Intelligence as an Experimental Science.* Technical Report KSL 87-03, Knowledge Systems Laboratory, Stanford University, January 1987.

[5] William J. Clancey. Classification problem solving. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, page 49, 1984.

[6] Paul R. Cohen, Alvah Davis, David S. Day, Michael Greenberg, Rick Kjeldsen, Sue Lander, and Cindy Loiselle. Representativeness and uncertainty in classification systems. *AI Magazine*, 6(3):136–149, Fall 1985.

[7] Paul R. Cohen and Rick Kjeldsen. Information retrieval by constrained spreading activation in semantic networks. *Information Processing and Management*, 23(4):255–268, 1987.

[8] Paul R. Cohen and Mark D. Lieberman. Folio: An expert assistant for portfolio managers. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 212–215, 1983.

[9] Geraldine Dawson, Charles Finley, Sheila Phillips, and Larry Galpert. Hemispheric specialization and the language abilities of autistic children. *Child Development*, 57(6):1442–1444, December 1986.

[10] J.R. Dixon, M.K. Simmons, and P.R. Cohen. An architecture for application of artificial intelligence to design. In *Proceedings of ACM/IEEE 21st Design Automation Conference*, Albuquerque, NM, 1984.

[11] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy. The HEARSAY-II speech understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, 12:213–253, 1980.

[12] John Gaschnig, Philip Klahr, Harry Pople, Edward H. Shortliffe, and Allan Terry. Evaluation of expert systems: Issues and case studies. In F. Hayes-Roth, D. A. Waterman, and Douglas B. Lenat, editors, *Building Expert Systems*, chapter 8, pages 241–280, Addison-Wesley, 1983.

[13] James R. Geissman and Roger D. Schultz. Verification and validation. *AI Expert*, 3(2):26–33, February 1988.

[14] M. Henrion and D.R. Cooley. An experimental comparison of knowledge engineering for expert systems and for decision analysis. In *Proceedings of the Sixth National Conference on Artificial Intelligence, AAAI*, 1987.

[15] F.S. Hillier and G.J. Lieberman. *Introduction to Operations Research*. Holden-Day, Inc., San Francisco, CA, 1980.

[16] A. Howe, J.R. Dixon, P.R. Cohen, and M.K. Simmons. Dominic: A domain-independent program for mechanical engineering design. *International Journal for Artificial Intelligence in Engineering*, 1(1):23–29, July 1986.

[17] Adele Howe. Learning to design mechanical engineering problems. 1986. EKSL Working Paper 86-01, University of Massachusetts.

[18] John G. Kemeny. *A Philosopher Looks at Science*. Van Nostrand Reinhold Co., New York, 1959.

[19] Pat Langley. Research papers in machine learning. *Machine Learning*, 2(3):195–198, November 1987.

[20] D.B. Lenat. *AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search*. PhD thesis, Department of Computer Science report STAN-CS-76-570, Stanford University, 1976. Doctoral Dissertation.

[21] T. M. Mitchell. Version spaces: a candidate elimination approach to rule learning. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 305–310, Tiblisi, Georgia, USSR, 1977.

[22] T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli. Explanation-based generalization: a unifying view. *Machine Learning*, 1(1):46–80, 1986.

[23] Ulric Neisser. *Cognition and Reality*. Freeman Press, San Francisco, 1976.

[24] A. Newell. A tutorial on speech understanding systems. In D. R. Reddy, editor, *Speech Recognition: Invited Papers Presented at the 1974 IEEE Symposium*, pages 3–54, Academic Press, New York, 1975.

[25] A. Newell. You can't play 20 questions with nature and win. In W. G. Chase, editor, *Visual Information Processing*, pages 283–308, Academic Press, NY, 1973.

[26] Allen Newell and Herbert A. Simon. Computer science as empirical inquiry: symbols and search. *Communications of the ACM*, 19(3):113–126, March 1976.

[27] Mark Ferral Orelup. *Meta-Control in Domain-Independent Design by Iterative Redesign.* Master's thesis, Mechanical Engineering Department, University of Massachusetts, September 1987.

[28] C. J. Van Rijsbergen. *Information Retrieval.* Butterworths, London, second edition, 1979.

[29] Jeff Rothenberg, Jody Paul, Iris Kameny, James R. Kipps, and Marcy Swenson. *Evaluating Expert System Tools: A Framework and Methodology.* Technical Report R-3542-DARPA, Rand Corporation, July 1987.

[30] E. Shortliffe. *Computer-Based Medical Consultations: MYCIN.* American Elsevier: New York, NY, 1976.

[31] D. Waltz. *Generating Semantic Descriptions from Drawings of Scenes with Shadows*, pages 19–92. McGraw-Hill, New York, 1975.

## A. Appendix: Evaluation Criteria

The following tables summarize the evaluation criteria appropriate for each stage in the empirical AI research cycle.

1. Is the task significant? Why?
   (a) If the problem has been previously defined, how is your reformulation an improvement?
2. Is your research likely to contribute meaningfully to the problem? Is the task tractable?
3. As the task becomes more specifically defined for your research, is it still representative of a class of tasks?
4. Have any interesting aspects been abstracted away or simplified?
   (a) If the problem has been previously defined, have any aspects extant in the earlier definition been abstracted out or simplified?
5. What are the subgoals of the research? What key research tasks will be/have been addressed and solved as part of the project?
6. How do you know when you have successfully demonstrated a solution to the task? Is the task one in which a solution can be demonstrated?

Table 1: Criteria for evaluating research problems

1. How is the method an improvement over existing technologies?
   (a) Does it account for more situations? (input)
   (b) Does it produce a wider variety of desired behaviors? (output)
   (c) Is the method expected to be more efficient? (space, solution time, development time, etc.)
   (d) Does it hold more promise for further development? (for example, due to the opening up of a new paradigm)
2. Is there a recognized metric for evaluating the performance of your method? (e.g., normative, cognitively valid, etc.)
3. Does it rely on other methods? (Does it require input in a particular form or preprocessed? Does it require access to a certain type of knowledge base or routines?)
4. What are the underlying assumptions? (known limitations, scope of expected input, scope of desired output, expected performance criteria, etc.)
5. What is the scope of the method?
   (a) How extendible is it? Will it easily scale up to a larger knowledge base?
   (b) Does it address exactly the task? portions of the task? a class of tasks?
   (c) Could it, or parts of it, be applied to other problems?
   (d) Does it transfer to more complicated problems? (perhaps more knowledge intensive or more/less constrained or with more complex interactions)
6. When it cannot provide a good solution, does it do nothing or does it provide bad solutions or does it provide the best solution given the available resources?
7. How well is the method understood?
   (a) Why does it work?
   (b) Under what circumstances, won't it work?
   (c) Are the limitations of the method inherent or simply not yet addressed?
   (d) Have the design decisions been justified?
8. What is the relationship between the problem and the method? Why does it work for this task?

Table 2: Criteria for evaluating methods

1. How demonstrative is the program?
   (a) Can we evaluate its external behavior?
   (b) How transparent is it? Can we evaluate its internal behavior?
   (c) Can the class of capabilities necessary for the task be demonstrated by a well-defined set of test cases?
   (d) How many test cases does it demonstrate?
2. Is it specially tuned for a particular example?
3. How well does the program implement the method?
   (a) Can you determine the program's limitations?
   (b) Have parts been left out or kludged? Why and to what effect?
   (c) Has implementation forced a more detailed definition or even re-evaluation of the method? How was that accomplished?
4. Is the program's performance predictable?

Table 3: Criteria for evaluating method implementation

1. How many examples can be demonstrated?
   (a) Are they qualitatively different?
   (b) Do these examples illustrate all the capabilities that are claimed? Do they illustrate limitations?
   (c) Is the number of examples sufficient to justify the inductive generalizations?
2. Should program performance be compared to some standard? its tuned performance? other programs? people (cognitive validity)? experts and novices (expert performance)? normative behavior? outcomes? (either from the real world or from simulations)
3. What are the criteria for good performance? Who defines the criteria?
4. If the program purports to be general (domain-independent),
   (a) Can it be tested on several domains?
   (b) Are the domains qualitatively different?
   (c) Do they represent the class of domains?
   (d) Should performance in the initial domain be compared to performance in other domains? (Do you expect that the program is tuned to perform best in domain(s) used for debugging?)
   (e) Is the set of domains sufficient to justify inductive generalization?
5. If a series of related programs is being evaluated,
   (a) Can you determine how differences in the programs are manifested as differences in behavior?
   (b) If the method was implemented differently in each program in the series, how were these differences related to the generalizations?
   (c) Were difficulties encountered in implementing the method in other programs?

Table 4: Criteria for evaluating experiment design

1. How did program performance compare to its selected standard? (e.g., other programs, people, normative behavior, etc.)
2. Is the program's performance different from predictions of how the method should perform?
3. How efficient is the program? time/space? knowledge requirements?
4. Did the program demonstrate good performance?
5. Did you learn what you wanted from the program and experiments?
6. Is it easy for the intended users to understand?
7. Can you define the program's performance limitations?
8. Do you understand why the program works or doesn't work?
   (a) What is the impact of changing the program even slightly?
   (b) Does it perform as expected on examples not used for debugging?
   (c) Can the effect of different control strategies be determined?
   (d) How does the program respond if input is rearranged, more noisy, or missing?
   (e) What is the relationship between characteristics of the test problems and performance (either external or internal if program traces are available?)
   (f) Can the understanding of this program be generalized to the method? to characteristics of the method? to a larger class of tasks?

Table 5: Criteria for evaluating what the experiments told us