RADC-TR-89-347
Final Technical Report
February 1990

THEORY OF ENDORSEMENTS AND REASONING WITH UNCERTAINTY

University of Massachusetts

Sponsored by
Defense Advanced Research Projects Agency
ARPA Order No. 5294

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Services (NTIS) At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-89-347 has been reviewed and is approved for publication.

APPROVED: Chuian-Chuian Hwong

CHUIAN-CHUIAN HWONG
Project Engineer

APPROVED: Raymond P. Urtz Jr.

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: Igor G. Plonisch

IGOR G. PLONISCH
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COES) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

THEORY OF ENDORSEMENTS AND REASONING WITH UNCERTAINTY

Paul R. Cohen
David Day
Jeff Delisio
Mike Greenberg
Thomas Gruber
David Hart
Adele Howe
Cynthia Loiselle

Contractor: University of Massachusetts
Contract Number: F30602-85-C-0014
Effective Date of Contract: 23 January 1985
Contract Expiration Date: 30 June 1989
Short Title of Work: Theory of Endorsements and Reasoning with Uncertainty
Program Code Number: 9E20
Period of Work Covered: Feb 86 - Feb 89

Principal Investigator: Paul R. Cohen
Phone: (413) 545-3613

RADC Project Engineer: Chuian-Chuian Hwong
Phone: (315) 330-7794

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by Chuian-Chuian Hwong, RADC (COES), Griffiss AFB NY 13441-5700 under Contract F30602-85-C-0014.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-89-347
6a. NAME OF PERFORMING ORGANIZATION University of Massachusetts 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COES)
6c. ADDRESS (City, State, and ZIP Code) Lederle Graduate Research Center P O Box 571 Amherst MA 01003	7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Defense Advanced Research Projects Agency 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-85-C-0014
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Blvd Arlington VA 22209-2308	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 62301E; PROJECT NO. E294; TASK NO 00; WORK UNIT ACCESSION NO 01

11. TITLE (Include Security Classification)

THEORY OF ENDORSEMENTS AND REASONING WITH UNCERTAINTY

12. PERSONAL AUTHOR(S) Paul R. Cohen, David Day, Jeff Delisio, Mike Greenberg, Thomas Gruber, David Hart, Adele Howe, Cynthia Loiselle

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM Feb 86 TO Feb 89	February 1990	257

16. SUPPLEMENTARY NOTATION

N/A

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Knowledge acquisition; Control knowledge; Plausible reasoning; Knowledge systems; AI research methodology
12	05		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The general knowledge acquisition problem and the problem of acquiring strategic knowledge from experts was addressed. An automated knowledge acquisition tool called ASK was described and demonstrated with a human machine dialog, and the results from experiments analyzed. The importance of the design of knowledge representations and reasoning methods was emphasized since it plays a central role in the knowledge acquisition process.

Reasoning under uncertainty has two aspects. One is to assess the most likely states of the world, the other is to act on those assessments. The former is often called judgement and the latter decision-making. Judgement has been the primary focus of research on reasoning under uncertainty in AI, while decision-making (lately these have been called planning problems) which deals with how autonomous agents act in uncertain environments, is increasingly gaining more attention. An adaptive planner called PLASTYC was built to operate in a dynamic, spatially-distributed, multi-agent, —
(Continued on Reverse)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED. ☒ SAME AS RPT. ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Chuian-Chuian Hwong	22b. TELEPHONE (Include Area Code) (315) 330-7794
	22c. OFFICE SYMBOL RADC (COES)

DD Form 1473, JUN 86 — Previous editions are obsolete.

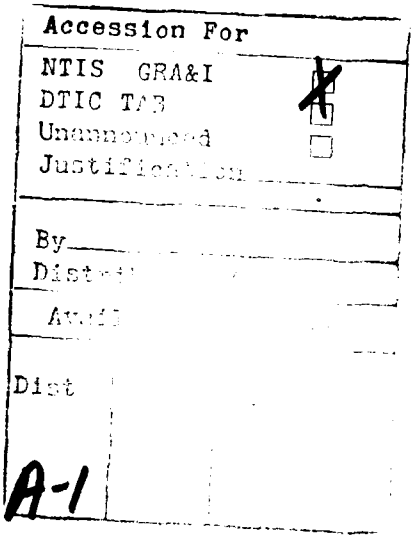
SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Block 19 continued:

ongoing, unpredictable, and real-time world simulator for controlling forest fires. The simulator, called Phoenix, was used as a framework to discover functional relationships between environment characteristics, autonomous agents' behaviors, and agents' designs.

A knowledge system that finds sources of funding for research proposals called GRANT was used to study the deep structures for plausible inference rules in semantic networks. Plausible inference rules can be automatically derived from the relations in knowledge bases and judgements of plausibility for the conclusions of these rules predicted. But further work is required to prove the generality of the results.

The roles of evaluation in empirical AI research are described with case studies. Five classes of evaluation criteria appropriate for each stage in the empirical AI research cycle are presented.



Contents

Chapter 1

Overview of the Research

Figure 1.1 gives an overview of the projects and the important links between them over the history of the Experimental Knowledge System Laboratory (EKSL) at the University of Massachusetts. Those projects surrounded in grey are funded by this contract and are discussed in detail in the following chapters[1]. This introduction gives a historical overview of the projects and their interconnections; it places the chapters that follow in their proper research perspective.

1.1 Plausible inference

A premise of our work on endorsements was that degrees of belief are ambiguous—one rarely knows what a person means by "my degree of belief is 0.7." Around 1985, we became concerned that endorsements in SOLOMON were also ambiguous. Endorsements had evocative names like **the-rule-might-be-too-general** but what did they really *mean*? The rules to assert and modify endorsements in HMMM were one attempt to specify their semantics, but we still worried a lot about the issue (although today it seems relatively unimportant). We decided to build an endorsement-based system in a domain where the causes of uncertainty, and thus the interpretations of endorsements, were clear.

We selected an information retrieval problem in which uncertainty was due to disparities between queries and retrieved entities. The problem was to find one or more agencies to fund a research proposal, given the topics in the proposal and the research goals of many agencies. One's certainty that an agency will fund a proposal is proportional to the *degree of fit* between the agency's goals and the researcher's topics, respectively. (Degree of fit, or representativeness, is a well-understood interpretation of uncertainty judgments, and is common in AI tasks such as classification.) Thus, endorsements would be interpreted as reasons to believe the degree of fit was either high or low, and would reflect different ways that goals and topics could fit or not fit [Cohen *et al.*, 1985]. For example, if an agency wants to fund research on **tobacco**, and a researcher proposes to study **nicotine**, is this a fit or not? What if the agency wants to support research on **nicotine** and the researcher proposes only to study **tobacco**? Intuitively, the fit seems better in the first case: If an agency supports work on **x**, then it may support work on **y** if **y part-of x** but not if **y has-part x**. The degree of fit between two concepts is a function of the relationship that holds between

[1]The papers that these chapters are drawn from are listed in the appendix.

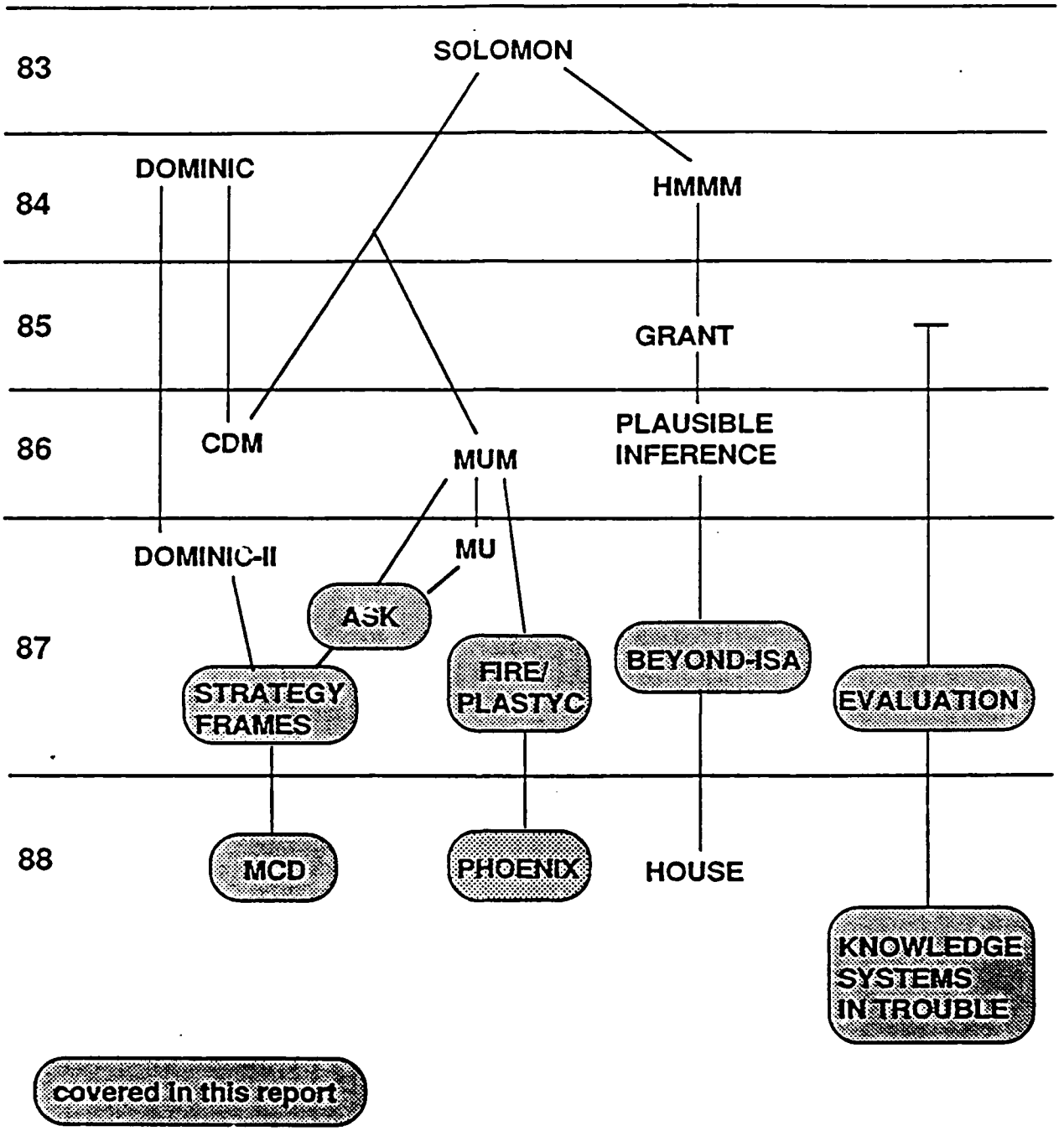


Figure 1.1: The projects of the Experimental Knowledge Systems Laboratory at the University of Massachusetts from 1983 through 1988.

them in a semantic net, and endorsements simply denote these relationships (and also chains of relationships, such as multiple steps up an isa hierarchy).

The GRANT system found funding agencies for proposals by finding "good" chains of relationships between them in a semantic network. The network was, in effect, a large index for funding agencies, each of which was associated with the nodes in the network that represented its interests. Proposals would activate nodes, and the activation would spread until it led to agencies. Some pathways indicate that a proposal topic is a good fit to an agency goal, others indicate the opposite. This search, which we called constrained spreading activation, was remarkably good at finding the most appropriate agencies: An early version of GRANT, with 50 agencies and 800 nodes in the network, had a hit rate of 80% and a false-positive rate of 32% [Cohen *et al.*, 1985]. Later and larger versions performed less well [Cohen and Stanhope, 1986],[Kjeldsen and Cohen, 1987], though still better than naive keyword methods and on par with statistical methods. (See chapter 7 and[Cohen and Kjeldsen, 1987] for details on how we evaluated GRANT.)

In another application of GRANT's method of constrained spreading activation to IR [Croft *et al.*, 1988] we combined it with a citation based search technique, resulting in better performance than with either technique used seperately.

Very recently, we generalized the spreading activation method from GRANT to a kind of plausible inference. We showed how to automatically derive plausible inference rules from the relationships in a knowledge base, generate thousands of inferences, and differentiate plausible conclusions from implausible ones (see chapter 6). We generated 315 rules of inference by a simple syntactic method, then instantiated them with concepts from the GRANT knowledge base to generate over 3000 inferences. Human subjects judged their plausibility. This experiment told us which rules generate plausible conclusions, but not why. A second experiment strongly supported the hypothesis that rules with transitive deep structures are generally plausible while intransitive rules are not. This result has major implications for building large, common sense knowledge bases, because if we can generate rules of plausible inference from the relationships in a knowledge base, and automatically predict whether humans will find their conclusions plausible (as our results strongly suggest) then we can speed the construction of knowledge bases by automatically adding hundreds of thousands of plausible conclusions. Cynthia Loiselle is currently refining and using these methods to build a very large knowledge base as part of her dissertation research.

1.2 Decision-making

During the HMMM and GRANT projects, we worked with Prof. Jack Dixon in the Department of Mechanical Engineering on a domain-independent architecture for mechanical design [Dixon *et al.*, 1984]. Although this system, called DOMINIC, wasn't part of our research on uncertainty, it has strongly influenced subsequent projects (see Fig. 1.1). DOMINIC's strategy was iterative refinement, implemented as hill-climbing. States in its search space were designs, and the operators each modified an aspect of the current design—if this improved the design overall [Howe *et al.*, 1986]. We came to view decision-making in the same terms: as a constructive process in which the decision is iteratively refined by the decision-maker. We developed a 24-state search space that was sufficient to model the construction of two-alternative, multi-attribute decisions. This became the core of a decision-support system called CDM, for constructive decision maker [Howe and Cohen, 1987],[Howe and Cohen, 1988]. To our knowledge, CDM is the first system to automate the process of constructing decisions, which is typically done by human decision analysts.

CDM inherited from SOLOMON the view that representations of uncertainty should guide problem-solving actions. It implemented that view as state-space search. Every state was vector of five parameters. Depending on its value, each parameter was a reason to either accept a decision or acquire more evidence. Constructing a decision was just searching for a state in which a decision could be accepted. While conceptually simple, CDM led to three conclusions that have influenced all our subsequent work. First, reasoning under uncertainty involves tradeoffs between certainty and resources such as time, money, and computation: CDM's search can be terminated (and a decision taken) whenever the anticipated value of increased certainty isn't worth the cost of evidence and processing evidence. Second, because evidence by definition has unknown effects on one's certainty, one cannot *plan* a sequence of actions in the traditional sense of projecting the known effects of operators: Each of CDM's decision states had at least two successors, one more certain and one less certain, respectively, and CDM did almost no lookahead in its space of decision states. In fact, the way CDM constructed decisions anticipated current work on reactive planning. Third, and more generally, CDM convinced us that AI approaches to uncertainty must have associated decision procedures, that is, mechanisms to produce, for any decision state, one or more problem-solving actions [Cohen, 1986].

Two questions are raised by the last conclusion: First, by what standards

should we judge the quality of decisions? Decision theory is normative but may require too much information to be useful in AI and is better suited to static "one shot" decisions than ongoing problem-solving (see chapter 4 and[Cohen, 1987a] for discussions of the implications of these arguments). We based our subsequent research on two other standards: systems should make the same decisions as experts (see Sec. 1.3) and systems should satisfice under resource limitations (see Sec. 1.4). The second question is, what computational metaphors (and ultimately, implementations) do we want for decision procedures? In CDM, we viewed decision-making in terms of search; in subsequent work, we viewed it in terms of control and, most recently, planning.

1.3 Reasoning under uncertainty viewed as control

Around 1986, we came to view reasoning under uncertainty as a control problem; that is, we argued that the influence of uncertainty on the behavior of AI systems would be felt in their control strategies [Cohen, 1986],[Cohen, 1987b]. We surveyed the literature on control from this perspective [Cohen, 1987a] and became convinced that some of the most successful examples of reasoning under uncertainty (e.g., HEARSAY-II, NEOMYCIN, MUM) relied on control strategies to "do the right thing" in uncertain situations.

At the time, this view ran afoul of the dominant approaches to uncertainty and knowledge-based systems, though it is now gaining momentum. Few AI tasks are *formulated* to require sophisticated control strategies to manage uncertainty [Cohen and Gruber, 1985]. Instead, they are formulated categorically (ignoring uncertainty) or so that reasoning under uncertainty means little more than calculating probabilities. We believed this is not for a lack of strategies for managing uncertainty, but for a lack of explicit *representations* for these strategies. Given the representations, we expected that many uncertain problems could and would be solved without formulating them probabilistically. This proved correct: The last three years have seen increasing activity in the uncertainty and planning communities to develop representations for strategies; and the most recent AAAI Uncertainty Workshop closed with near consensus that reasoning under uncertainty has a significant control aspect. One impediment has been the prevailing view that the power of knowledge-based systems derives from "substantive" knowledge

(which is usually explicit), not from how the knowledge is used. Control is regarded as unimportant and relegated to the interpreter. In contrast, our view is that control knowledge is part of the expertise of a domain. It is "how to" knowledge (specifically, how to act in uncertain situations) and should be represented explicitly for the same reasons that we represent other kinds of domain expertise explicitly. (The ramifications of this point are explored in three papers on knowledge engineering by Cohen and Gruber [Gruber and Cohen, 1987b],[Gruber and Cohen, 1987a],[Gruber and Cohen, 1987c].)

About this time, we began to collaborate with an internist, Dr. Paul Berman. He introduced a distinction between retrospective diagnosis, in which the physician has all the potentially relevant evidence and is "looking back" to see what it means; and prospective diagnosis, in which the physician is "looking forward," or planning, the diagnostic workup. Retrospective diagnosis systems generally do not have sophisticated control strategies. In retrospective diagnosis, the subcomponents of control—focus of attention, control of inferences, and control of actions or questions—are either trivial or nonexistent. To explore the view that uncertain problems can be solved by formulating them in terms of experts' strategies, we decided to build a *prospective* diagnosis system with explicit, expert strategies for controlling questions, tests, and treatments. To explore the role of probability (if any) in the system, we provided it with crude degrees of belief [Gruber and Cohen, 1987c],[Cohen *et al.*, 1987c].

The MUM system (for managing uncertainty in medicine) constructed workups for chest pain [Cohen *et al.*, 1987a]. It was moderately successful at resolving one of the most difficult differentials in internal medicine, atypical angina and esophageal spasm. It produced expert workups; that is, it asked the same questions as an expert, in the same order, for the same reasons. It supported the conclusion that reasoning under uncertainty is a control problem, and it provided these insights about the role of probability: First, probabilities serve control. MUM's degrees of belief in hypotheses, the prior probabilities of hypotheses, and the conditional probabilities given evidence that might be collected in future, all contributed to MUM's decisions about what to do next. Second, calculating probabilities is not itself a strategy for solving problems under uncertainty (however much it appears to be in retrospective diagnosis), but it does support these strategies.

While MUM was being built, we were confronted with two problems that, ultimately, had a common solution. First, we wanted a more general result than MUM: we wanted an *architecture* that incorporated strategies for prospective diagnosis. Second, we were dissatisfied that it often

took hours to implement simple strategies in MUM. We became influenced by Chandrasekaran's view that tasks such as diagnosis and design could be described at a general or "generic" level, and by Clancey's HERACLES architecture for classification problem solving. We coined the term *task-level architecture* to emphasize that these architectures were in effect virtual machines for particular tasks, independent of how they were implemented [Gruber and Cohen, 1987b]. An expert ought to be able to prescribe control strategies *in terms of the task*, without thinking about the implementation of these strategies. With this orientation, researchers in John McDermott's group had been able to acquire specific strategies by asking experts to specialize a weak, general strategy for construction tasks; so we hoped we might be able to speed the acquisition of diagnostic strategies by asking our expert to specialize a weak strategy for prospective diagnosis. For these reasons—the desire for a general architecture and faster knowledge acquisition—the next step was clearly to build a task level architecture for prospective diagnosis.

MU is such an architecture [Cohen *et al.*, 1987b]. It incorporates this weak diagnostic strategy:

1. Select a focus of attention
2. Select a question to ask
3. Ask the question
4. Propagate the answer to the question through long-term memory, changing degrees of belief in hypotheses as appropriate
5. Go to 1.

Foci of attention and questions are determined by control rules (similar to Davis' meta-rules). The conditions of these rules contain *control features*, which are the reasons an expert gives for his actions. MU supported the definition, maintenance, and access to control features, and thus the definition of control rules like this one (from [Gruber, 1988a]):

> Select ?action if:
> gather-evidence-for-differential is a **goal**, and
> ?hypothesis is in **the-differential**, and
> ?action **potentially-confirms** ?hypothesis, and
> the **cost** of ?action $\leq$ low, and
> the **time-required** of ?action = few-minutes.

The interpretation of control features is that they define sets. The conditions in control rules test membership in these sets. For example, in the previous rule, the set of goals must include gather-evidence-for-differential;

?hypothesis must be in the set called the-differential; and ?action must simultaneously belong to three sets of actions: those that potentially confirm ?hypothesis, those whose cost is $\leq$ low, and those that take a few minutes.

Using MU, Tom Gruber built a system called ASK (for acquiring strategic knowledge) that acquires rules like these from experts (see section 2). ASK acquires strategic knowledge in the context of cases, that is, it runs MUM until the latter proposes an inappropriate action (a question, test, or treatment), at which point it acquires rules from the expert to generate the correct action. It also automatically generalizes the rules by simple induction techniques to other situations in which classes of actions are appropriate and inappropriate [Gruber, 1988a],[Gruber, 1987].

MU and ASK culminate the research on control that we began with SOLOMON. Common to these projects is the view that explicit representations of uncertain situations (called endorsements in SOLOMON and control features in MU and ASK) affect problem-solving behavior. The value of MU and ASK is not so much that they they reinforced this point, but that they showed us how to build systems like MUM that act appropriately in uncertain situations: One begins with a task-level architecture in which control is determined by explicit control features. Because control features are the *reasons* an expert gives for actions, they are a good medium for acquiring expert diagnostic strategies. They not only determine control of reasoning under uncertainty, they justify it, and thus help acquire expertise about how to do it.

1.4 Reasoning under uncertainty viewed as planning

A disturbing aspect of MUM and MU is that control is very local: no history or projection is used to select actions. While it appeared that Dr. Berman, MUM's expert, planned two or three actions at a time, MUM itself simply selected the best action given the current state as summarized by control features. MUM was a reactive planner. Reactive planning is an extreme response to uncertainty: its premise is that we cannot accurately project the effects of actions in an uncertain future, but its conclusion is that we shouldn't try. We think the premise is too categorical. Even when it's true, the conclusion doesn't follow, because even inaccurate projections can be useful.

The purposes of projection are to avoid pitfalls or dead-ends between

the current state and a search horizon, and to increase the efficiency of a solution by finding the cheapest path between the current state and a search horizon (chapter 4 and [Cohen and Day, 1987]). (When backtracking is allowed, pitfalls are just a kind of inefficiency. In the domains that interest us, especially real-time domains as discussed below, backtracking isn't allowed). But in domains like MUM's, reactive planning is quite adequate: few pitfalls lurk in the workup of chest pain, and inefficiency can be avoided by control rules that prevent MUM asking questions that might initiate inefficient sequences of questions.

In this sense, we had been lucky with MUM, but in another sense, the domain was impeding progress. First, we wanted to build a planner that somehow slipped between two contradictory positions: Projection is impossible because the future is uncertain, yet pitfalls and inefficiencies must be avoided by projection. Second, with ASK months from completion, we were spending too much time learning medicine and too little time acquiring Dr. Berman's diagnostic strategies. Our main interest was *how* to solve uncertain problems, so we wanted to work in a game-like domain where strategy dominates substantive knowledge. Third we had become interested in real-time problems, and resource-limited problems in general, where we could assess the value of evidence in terms of its cost in resources.

During the summer of 1987, we built a large simulator of forest fires (see Fig. 1.2) and the equipment commonly used to put them out. The progress of the fire depends on ground cover, fuel moisture, slope, wind speed and direction, rivers, roads, and the actions of bulldozers, crews, planes and helicopters. The latter objects are controlled by a planner—initially a human, but recently a hybrid reactive/projective planner called PLASTYC. David Day has developed PLASTYC for his Ph.D. PLASTYC's task is challenging because:

- Knowledge of the fire is limited to what the agents in the field (crews, bulldozers, etc.) can "see." The planner rarely, if ever, has complete knowledge of the extent or location of the fire (see Fig. 1.2).
- The behavior of the fire cannot be accurately predicted because some factors that affect it (e.g., terrain, ground cover and the moisture content of the ground cover) are known only approximately. Moreover, wind speed and direction can change unpredictably.
- The behavior of the fire-fighting agents cannot be accurately predicted. In particular, the time required to move to a location or perform some

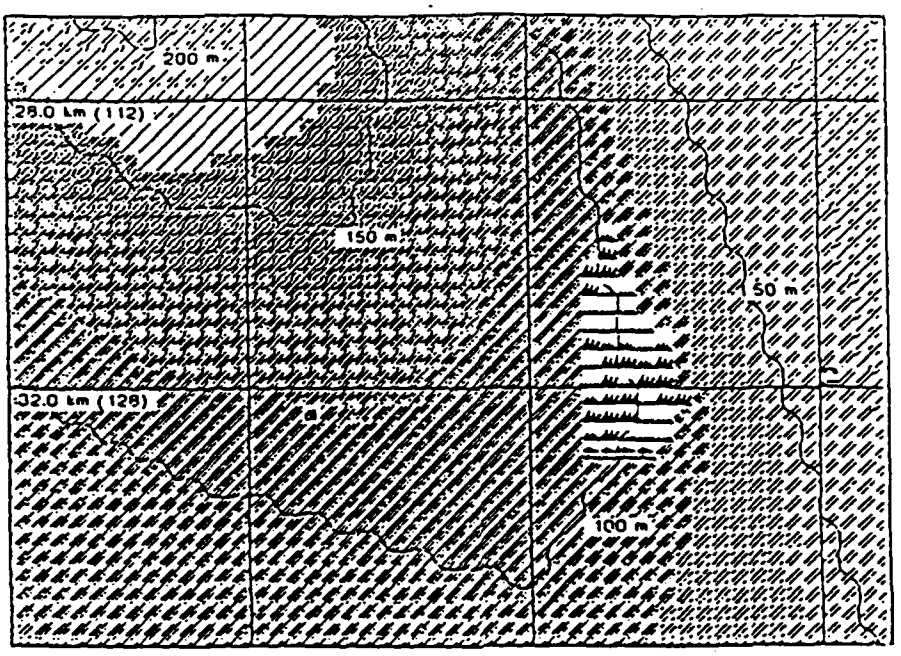
task depends on terrain and ground cover. Fire-fighting agents also have limited autonomy (e.g., to run away from a fire) so the planner cannot always be sure of their location.

- The simulation is real-time with respect to the fire. While fire-fighting agents move, cut line and drop retardent, the fire keeps burning. Most important, any time the planner devotes to deliberation is claimed as real estate by the fire.

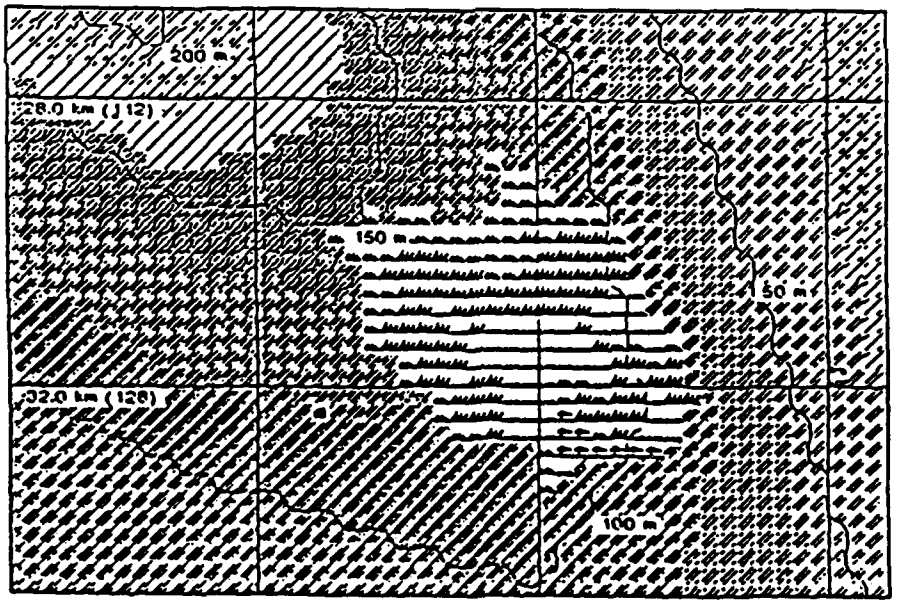
To plan in this environment, PLASTYC needs to balance projection and reaction. Projection is desirable to avoid pitfalls and increase efficiency. But it takes time, and uncertainty precludes avoiding *all* pitfalls, and efficient plans are useless if no time remains to execute them. The timeliness of reactions often makes the difference between success and failure.

PLASTYC is not complete, so our conclusions from it are primarily methodological at this point. We are convinced (and argued at length in chapter 4) that strategies for reasoning under uncertainty—their representation, organization, acquisition, and interpretation—should be studied in the context of dynamic, ongoing, resource-limited environments in which the ratio of strategies to substantive knowledge is high, and one can automatically generate many test cases. Indeed, we have initiated two other projects around environments with these characteristics. One is a process-control problem, in which a planner must keep a simulated fast-food restaurant from developing large shortages or surpluses, despite uncertainty about future fluctuations in demand. This project resulted in a richer declarative representation of control strategies than MUM's control rules (see chapter 3). The other is a planner for market strategy, in which the environment is the computer market (this is used by DEC; see the section on Technology Transfer).

An exciting substantive conclusion of this work is that the distinction between reactive and projective planning is eroding, and that this implies a role for adaptation. Some features of the forest fire are rough predictors of how the fire will evolve, so it is possible to create the ***illusion*** of projective planning by reacting to these predictors. It seems that humans and other animals accomplish many tasks this way, by simply reacting to configurations in the environment that are associated with future outcomes. Some of these reactions evolve, others are learned by individuals, and in MUM they are acquired from an expert by ASK. So at this point in our research, the big question is:



(A)



(B)

Figure 1.2: The state of the fire as seen by the planner (a), and the state of the fire as it really is (b).

How do intelligent agents adapt to uncertain environments?

One of the critical issues here is the evaluation of the contribution of the particular components of an agent to its performance. We want to know not only how to create an agent that performs well, but also why it performs well. Issues of evaluation are addressed in chapters 7 and 8. The forest-fire simulator (chapter 5) provides us with a testbed for exploring these issues. It gives us a complex, real time, environment where issues of uncertainly can not be avoided.

Currently we are trying to make the forest-fire planner learn from its experiences with the fire. We are looking at ephemeral adaptation or adjustment to fire conditions, and also at permanent changes in the knowledge and control strategies of the planner. This work, described in prospect in [Cohen, 1988], is the basis of Adele Howe's PhD research.

1.5 Technology Transfer

The technology that we have developed during this contract has started to be transfered in several ways. We are currently a likely candidate for the AFOSR Intelligent Real Time Problem Solving Initiative; if Phoenix is selected all of the contractors will use it.

During FY 1990 Gerald Powell from CECOM will be working with us on learning and planning in Phoenix. Powell is a Secretary of the Army Research and Study Fellow. This is an important opportunity to get our technology into DoD sites. This year, we been have working with Powell on adapting envelopes to the operational planning problem for the European theater. Next year we will apply these ideas in a real system.

We will initiate discussion with the SIMNET community during FY 1990 about getting Phoenix agents to participate as autonomous SIMNET nodes. This is an excellent opportunity to test our technology in an even more realistic environment, and provides an opportunity for considerable savings of manpower in SIMNET exercises.

Industry relations continue to be productive. We are working with Digital Equipment Corporation on a project very similar to the SIMNET idea. DEC is building a large, distributed simulation of the computer market, and running it to see how the market (and DEC's position within it) evolves. Phoenix agents will play the role of other companies, bringing out products, marketing, and executing various strategies to test DEC's long-term plans.

Part II

Control of Reasoning

Chapter 2

Automated Knowledge Acquisition for Strategic Information

2.1 Introduction

Knowledge acquisition is the transfer and transformation of knowledge from the forms in which it is available in the world into forms that can be used by a knowledge system (adapted from Buchanan et al., 1983). In the context of this article, knowledge in the world comes from people and knowledge in the system is implemented with formal symbol structures — knowledge representations. Knowledge acquisition is a multifaceted problem that encompasses many of the technical problems of knowledge engineering, the enterprise of building knowledge systems. Deciding what knowledge can be brought to bear for a problem, how the knowledge can be used by a program, how to represent it, and then eliciting it from people and encoding it in a knowledge base are all aspects of the knowledge acquisition problem. The inherent difficulty of these tasks make knowledge acquisition a fundamental obstacle to the widespread use of knowledge system technology.

The research reported here addresses the problem of acquiring strategic knowledge from people. In particular, the article presents an approach by which an interactive computer program assists with the knowledge acquisition process. The general term *automated knowledge acquisition* refers to computer-mediated elicitation and encoding of knowledge from people.

The first section of this article provides a theoretical analysis of the general knowledge acquisition problem and introduces the problem of acquiring strategic knowledge. Section 2.2 reviews the techniques of automated knowledge acquisition in terms of the theoretical framework developed in the first section, and motivates the present work. Section 2.3 describes the automated knowledge acquisition tool called ASK. Section 2.4 demonstrates the program with a human-computer dialog. Sections 2,5, 2.6, and 2.7 provide an analysis of the scope of applicability, assumptions, and limitations of the system, and a discussion of key design decisions. A concluding section summarizes the contribution of the design of knowledge representations to the development of knowledge acquisition tools.

2.1.1 The knowledge acquisition problem as representation mismatch

Most knowledge systems are built by knowledge engineers rather than by the domain experts who provide the knowledge. A long-standing goal of a course of knowledge acquisition research has been to replace the knowledge engineer with a program that assists in the direct "transfer of expertise" from experts to knowledge bases (Davis, 1976). Yet the problem has eluded a general solution; no existing knowledge acquisition program can build a knowledge system directly from experts' descriptions of what they do.

Why is knowledge acquisition difficult to automate? It seems that the "transfer" metaphor is misleading. Clearly, the form in which knowledge is available from people (e.g., descriptions in natural language) is different from the form in which knowledge is represented in knowledge systems. The difference between the two forms of knowledge, called *representation mismatch* (Buchanan

et al., 1983), is central to the problem of knowledge acquisition. Because of representation mismatch, one cannot merely transfer knowledge from human to machine. The knowledge acquisition tool must actively elicit knowledge in a form that can be obtained from domain experts, and map elicited knowledge into the executable representations of the knowledge system. The mapping is difficult to automate because the requirements for building a working system (e.g., operationality, consistency) differ from the requirements for a human expert describing a procedure to another person. In order to automate knowledge acquisition one must provide a method for overcoming representation mismatch.

The following discussion introduces three aspects of representation mismatch — modeling, operationalization, and generalization — as an explanatory framework with which to understand the problem of knowledge acquisition. The general issues and the specific problems of acquiring strategic knowledge are described within this framework.

Dimensions of representation mismatch

The *modeling* or formalization problem is a fundamental kind of representation mismatch. A knowledge system can be thought of as a qualitative model of systems in the world, including systems of intelligent activity (Clancey, 1987). While the model embodied by a knowledge system is informed by the behavior of human experts, it is not designed as a model of the experts' knowledge or their cognitive processes (Winograd & Flores, 1986). From this point of view, knowledge acquisition is a creative rather than imitative activity, resulting in a computational model that makes distinctions and abstractions not present in the initial language of the expert. Because of the difference between descriptions of behavior and computational models of action, the task of knowledge acquisition requires a model-building effort beyond that of rendering the expert's utterances in formal notation. Morik (1988) illustrates the modeling problem with the example of building a natural language understanding system. The builder of such a system does not interview experts in natural language understanding (native speakers) but experts in modeling the formal structure and mechanisms of language (linguists). Furthermore, the system-builder must adapt the expert's concepts (a theory of syntax) to the needs of a computational model (a parser), and sometimes invent new concepts (semantic networks).

The *operationalization* aspect of representation mismatch refers to the difference between descriptions of what the system should accomplish, given by domain experts, and the operational methods for achieving those objectives required by a computer program. Two senses of operationalization have been identified in the machine learning literature: making advice executable (Mostow, 1983) or more useful (Keller, 1988).[1] Knowledge acquisition involves both kinds of operationalization in the service of performance goals such as recommending an effective drug therapy or designing an efficient electric motor. To make a therapy recommendation executable, a knowledge engineer might build a interface that justifies a recommendation and requests the results. To make the advise "minimize cost, maximize speed" more useful, the engineer might decide to use a

[1]Dietterich and Bennett (1988) refer to "making goals achievable" and "making goals more useful."

redesign algorithm and elicit more knowledge from the expert about ways to cut costs and fine tune performance by modifying existing designs. The methods in which expert-supplied specifications are operationalized may require concepts and terminology unfamiliar to the domain expert.

A third dimension of representation mismatch is *generalization*: the difference between a set of specific examples of desired input/output performance and a more concise representation that will enable a system to perform correctly on a larger class of input situations. It is frequently observed that it is much easier to elicit examples of expert problem solving than general rules or procedures that cover the examples. The available form of knowledge (classified examples) needs to be mapped into a more useful representation (general class descriptions).

Problems of modeling, operationalization, and generalization are ubiquitous in knowledge acquisition. We will now see how they are manifest in the case of a particular kind of knowledge, strategic knowledge.

2.1.2 The problem of acquiring strategic knowledge

Strategic knowledge

Strategic knowledge is knowledge used by an agent to decide what action to perform next, where actions have consequences external to the agent. The more general term *control knowledge* refers to knowledge used to decide what to do next. What constitutes an action and its consequences depends on how one characterizes what the agent can *do*. For knowledge systems that make recommendations to people (e.g., "increase dosage of drug D") or control physical systems (e.g., "close valve V"), actions have consequences that are observable in the world outside of the agent. For problem-solving programs based on state-space search, an action may be the firing of a rule or an operator. For such an agent, *search-control knowledge* is used to choose internal actions that increase the likelihood of reaching a solution state and improve the speed of computation. The research reported here distinguishes knowledge for deciding among actions with consequences in the external world because the goal is to acquire strategic knowledge from domain experts without reference to the symbol-level organization of the knowledge system.

For descriptive purposes, strategic knowledge is also distinguished from the *substantive knowledge* of a domain, knowledge about what is believed to be true in the world. Both substantive and strategic knowledge underlie expertise in many domains. For example, a robot uses substantive knowledge to recognize and interpret situations in the world (e.g., an obstacle in its path) and strategic knowledge to decide what to do (to go around or over it). A lawyer uses substantive knowledge to identify the relevant features of cases and strategic knowledge to decide which case to cite in defense of an argument. A diagnostician uses substantive knowledge to evaluate evidence pro and con hypotheses, and uses strategic knowledge to decide among therapeutic actions. In general, substantive knowledge is used to identify relevant states of the world and strategic knowledge is used to evaluate the utility of possible actions given a state.

Representation mismatch for strategic knowledge

Although progress has been made in automating the acquisition of substantive knowledge used in classification (e. g., Bareiss, 1989; Boose & Bradshaw, 1987; Eshelman, 1988), strategic knowledge is typically imparted to systems by knowledge engineers using implementation-level mechanisms. The difficulty of acquiring strategic knowledge directly from experts can be seen within the framework of the three aspects of representation mismatch introduced earlier.

First, strategic knowledge presents serious modeling problems. While substantive knowledge might be acquired in a perspicuous form, such as rules mapping evidence to hypotheses, strategic knowledge about choosing actions is often represented with programming constructs, such as procedures or agenda mechanisms. At least in principle, rules that encode substantive knowledge can be written in a process-independent context; experts can specify how to classify situations in the world without worrying about the mechanism by which the specifications are interpreted. However, specifying knowledge that affects the order and choice of actions involves building a computational model of a process.

Consider the problem of modeling the strategy of a medical *workup*: the process of gathering data, assessing the results, and planning treatment for an individual patient. Although medical diagnosis is often described as a static classification problem (i.e. to classify *given* data), in medical practice evidence for a diagnosis is gathered over time, and the actions that produce evidence are chosen strategically. In modeling the workup, requests for patient data, laboratory tests, diagnostic procedures, and options for trial therapy are treated as actions. Substantive knowledge is used for the classification task, identifying likely causes for a given set of findings. In addition, strategic knowledge is used to decide what action to take next when the data are not all in.

In the MYCIN system, much of the knowledge that determined question ordering and decisions about laboratory tests was represented with screening clauses, clause ordering, and "certainty factor engineering" — implementation-level manipulations of the rules to achieve the intended strategic behavior (Clancey, 1983a). This knowledge could not be acquired easily with the available rule editors and debugging support tools (Buchanan and Shortliffe, 1984) because the strategy was implicit in the engineering tricks rather than the content of the rules. Since MYCIN, more explicit representations of strategic knowledge have been devised, such as the control blocks of S.1 (Erman, Scott, & London, 1984) and the high-level control languages of BB1 (Hayes-Roth, et al., 1987). Because these advances are general-purpose languages for control, rendering strategic knowledge in a computational model remains a programming task.

The acquisition of strategic knowledge also highlights the operationalization aspect of representation mismatch. At the *knowledge level* (Newell, 1982), the strategic knowledge of an agent may be specified as a set of behavioral goals that the agent should attempt to achieve. While it is possible to elicit specifications of desired behavior at the knowledge level from experts, it is far more difficult for experts (and knowledge engineers) to specify *how* a knowledge system should achieve these goals.

For example, during conventional knowledge acquisition for a knowledge system called MUM (Cohen et al., 1987), knowledge engineers interviewed a practicing physician for the purpose of modeling his diagnostic strategy for patients reporting chest and abdominal pain. MUM's task was to *generate* workups for chest pain patients, choosing one action at a time, waiting for the outcome of previous action. When asked to describe how to choose diagnostic tests, the expert would mention goals such as "do the cheap, quick tests first" and "protect the patient against a dangerous disease." This is nonoperational advice. To make it operational requires specifying how actions achieve goals (e.g., the diagnostic and therapeutic effect of actions), how to determine the currently relevant goals (e.g., when is a dangerous disease suspected), and how to balance competing objectives (e.g., cost, timeliness, diagnostic power, therapeutic value).

Third, the generalization aspect of representation mismatch is exhibited by the problem of acquiring strategic knowledge. By definition experts are good at what they do; it does not follow that they are good at generalizing what they do. In particular, it is much easier to elicit *cases* of strategic decisions — choices among actions in specific situations — than to elicit general strategies.

For example, in the MUM domain of chest pain workups, the physician makes a series of decisions about actions. He typically starts with a set of questions about patient history, then performs a physical examination (in a knowledge system, steps in the examination are also implemented as requests for data), and then plans and executes a series of diagnostic tests and trial therapeutic actions, until sufficient evidence for a conclusive diagnosis or recommended therapy has been found. For MUM it was feasible to elicit example workups corresponding to actual patients. These workups can be viewed as very specific plans. Each step in the workup, each choice of what to ask or try next, is the result of a strategic decision. However, generalizations about classes of strategic decisions were not present in the original workup descriptions but developed by retrospective analysis of the cases and follow-up consultation the expert. Within a single workup there may be several actions chosen for the same reasons (e.g., "do the cheap, quick tests first"), and there may be common reasons across workups (e.g., "gather enough evidence to recommend therapy").

Although cases of specific workups can be acquired in the form of directed graphs, they are not general enough for a knowledge system. First, they are specific to individual patients, and workups differ over individuals. Second, these plan-like procedures are extremely brittle; if any action can not be taken (e.g., because the results of a test are not available), then the procedures fail. Third, because they only record the results of strategic decisions, workup graphs fail to capture the underlying reasons for selecting actions in the prescribed order. This third problem reveals a subtle form of representation mismatch: although it is possible to elicit reasons for past strategic decisions, these reasons alone do not constitute a *generative* strategy. A generative strategy plans new workups based on the strategic knowledge that gave rise to existing workups.

The work reported in this article is motivated by the problem of acquiring knowledge that underlies strategic decisions and putting it in operational, general form. The next section lays out some of the techniques for addressing the problem.

2.2 Techniques for overcoming representation mismatch

Interactive tools can assist with knowledge acquisition by overcoming representation mismatch. This section reviews the techniques used by existing knowledge acquisition tools and motivates the approach taken in ASK. The techniques are presented in the context of the three aspects of representation mismatch.

2.2.1 Incorporating models into knowledge acquisition tools

Conventionally, the modeling problem for knowledge acquisition is handled by the knowledge engineer, who is responsible for building the knowledge system. The engineer analyzes the *performance task* (the problem to be solved by the knowledge system) and designs a program for applying knowledge to perform the task. A performance task is defined in terms of the input and output requirements of the system and the knowledge that is available. Tasks can be described at multiple levels of abstraction, from the functional specifications for a single application to input/output requirements for a general class of tasks. A *problem solving method* is the technique by which a knowledge system brings specific knowledge to bear on the task. When the computational requirements and methods for a *class* of tasks are well understood, a domain-independent problem-solving method can be designed, such as heuristic classification (Chandrasekaran, 1983; Clancey, 1985).

A *task-level architecture* consists of a knowledge representation language (a set of representational primitives) and a procedure implementing the problem-solving method designed to support knowledge systems for a class of performance tasks (Chandrasekaran, 1986; Gruber & Cohen, 1987). The procedure, which in this article is called the *method* for short, is a mechanism by which knowledge stated in the architecture's knowledge representation is applied to perform one of the tasks in the abstract class of tasks for which the architecture is designed. The representation and the method of a task-level architecture are tightly coupled. Each method defines *roles* for knowledge: ways in which knowledge is applied by the method (McDermott, 1988). The algorithm that implements the method in a program operates on statements in the associated representation language. The primitive terms in the representation correspond to the roles of knowledge. For example, Chandrasekaran (1987) and his colleagues have built architectures for *generic tasks* such as hierarchical classification and routine design. Each generic task is described in terms of the function to be performed (an abstract description of the performance task), a knowledge representation language (the set of primitive terms), and a "control strategy" (the procedure that implements the method). Chandrasekaran uses the term *generic task problem solvers* to refer to task-level architectures.

Task-level architectures can facilitate knowledge acquisition. Like a virtual machine, the architecture supports a set of method-specific representation primitives for building a knowledge system. Much of the model-building effort can be put into the design of the architecture, and the representational primitives can hide the implementation details. As a consequence the architecture can reduce representation mismatch by presenting a task-level representation language

comprehensible to the domain expert (Bylander and Chandrasekaran, 1987; Gruber & Cohen, 1987; Musen, 1989).

Interactive knowledge acquisition tools can help overcome representation mismatch by employing special techniques for eliciting and analyzing knowledge in architecture-supported representations. Some tools help analyze the task requirements to choose among existing methods and instantiate an architecture with domain terminology. For example, ROGET (Bennett, 1985) offers help in choosing among a small set of particular heuristic classification methods and elicits domain-specific instantiations of the input, output, and intermediate concepts for the selected method.

Other tools specialize in eliciting the knowledge for the roles required by the problem-solving method. For example, MOLE (Eshelman, 1988) uses a instantiation of the heuristic classification method called cover-and-differentiate. The knowledge acquisition tool specializes in the elicitation of knowledge for roles such as "covering knowledge" and "differentiating knowledge." Similarly, SALT (Marcus, 1988) is based on the propose-and-revise method for constructive problem solving, and elicits knowledge for proposing design extensions, identifying constraints, and backtracking from violated constraints.

Tools of another category specialize in a particular formulation of knowledge, independent of how the knowledge will be applied to particular tasks. For example, repertory grid tools elicit knowledge in the form of a two-dimensional matrix of weighted associations between "elements" and "traits" (Boose and Bradshaw, 1987; Shaw & Gaines, 1987). These tools use a task-independent elicitation technique to help the user identify traits and elements and the strengths of associations among them, and provides detailed analyses of the information. The user interprets the feedback in terms of a particular task, such as a procurement decision or an evaluation of policy alternatives.

On the other end of the spectrum are elicitation tools that are customized to the problem-solving method and a specific task in a domain. An example is OPAL, which acquires protocols used in the domain of cancer therapy (Musen, Fagan, Combs, & Shortliffe, 1987). The problem-solving method is a kind of skeletal-plan refinement, and the performance task is to manage cancer-therapy protocols modeled as skeletal plans. OPAL elicits knowledge from experts entirely in domain-specific terms, and in forms that correspond to paper and pencil representations familiar to the experts. Because the tool has almost completely eliminated the representation mismatch due to modeling, it has been used successfully by physicians with little experience with computation (Musen, 1989).

The acquisition of strategic knowledge, as it has been defined, is not supported by conventional task-level architectures. In fact, all of the built-in methods of the architectures mentioned above are implemented with procedures that themselves encode a control strategy. To the extent that the strategy is implemented by the method, it cannot be acquired by tools that assume the method is fixed.

However, it is possible to design an architecture for a restricted class of tasks that require domain-specific strategic knowledge. The method for such an architecture should define roles for strategic knowledge, just as MOLE's method defines roles for substantive knowledge, such as knowledge for proposing

explanations that cover an abnormal symptom. As will be described in Section 2.3, ASK was designed with an architecture that represents strategic knowledge as rules that map situations to desired actions. In this architecture, strategic knowledge is limited to three roles for associating features in the agent's current model of the world with classes of appropriate actions. As will be discussed in Section 2.6, the restricted roles for strategic knowledge reduces the scope of what needs to be acquired and simplifies how elicited knowledge is operationalized and generalized. It also limits the class of strategies that can be acquired.

2.2.2 Eliciting knowledge in operational terms

Automated knowledge acquisition tools can address the operationalization aspect of representation mismatch by limiting what is elicited from the user to representations of knowledge that are already machine-executable — that is, to elicit knowledge in the form in which it will be used for performance, or in some form that can be compiled into the runtime representation. An alternative approach is to provide a nonoperational "mediating representation" for eliciting the conceptual structure of a domain, and then *manually* building a system that operationalizes the specifications (Johnson & Tomlinson, 1988). A rule editor is a simple example of a tool that elicits knowledge in a form that can be directly executed.

The technique of eliciting knowledge directly in executable form is reminiscent of the single representation trick (Dietterich et al, 1982) in which the learning agent is given training data in the same representation as the language used for describing learned concepts. Using this technique in a knowledge acquisition tool replaces the problem of making the elicited input executable (operationalization) with the assumption that the elicitation language is representationally adequate. A language is representationally adequate if all of the relevant domain knowledge can be stated in the representation.

The success of tools employing this technique depends in part on whether the elicitation interface can make the operational semantics of the representation comprehensible to the user. For example, although TEIRESIAS paraphrases rules into English, the user needs to know more than English to understand them. TEIRESIAS depends on the assumption that the user can understand the backward-chaining model (Davis, 1976).

Well-designed user interface techniques can help make the computational model of the architecture comprehensible to the user. For example, the OPAL tool facilitates the acquisition of cancer treatment protocols with a form-filling interface, emulating paper-and-pencil forms familiar to its users (Musen et al., 1987). Similarly, spreadsheet applications are made comprehensible by presenting a familiar metaphor. The interface design goal is to minimize the conceptual distance between the user's understanding of the system's mechanism and the system's presentation of the options afforded by the computational model (Hutchins, Hollan, & Norman, 1986).

A tool that acquires knowledge in an executable representation can also offer intelligent assistance by analyzing the consequences of *applying* the knowledge. For example, SALT elicits fine-grained rules for repairing local constraint

violations in a design task. One of the consequences of using backtracking from local constraint violations is that the user can unintentionally define cycles in the dependency network, in which repairing one constraint violation introduces another. SALT can analyze the elicited knowledge, identify cycles, and offer assistance to the user in specifying different routes for backtracking (Marcus, 1987).

It is difficult to acquire strategic knowledge in executable form without forcing the expert to understand symbol-level mechanisms such as procedures and priority schemes. There is a tension between the requirement to provide the user with a language that is comprehensible and yet sufficiently powerful to implement the strategy. There are some techniques that help elicit specifications of control, such as visual programming interfaces for building transition networks (Musen Fagan, & Shortliffe, 1986) and graph-drawing tools for specifying decision trees (Hannan & Politakis, 1986). However, the strategic knowledge that can generate decisions among actions is *implicit* in transition networks and decision trees.

ASK's representation of strategic knowledge was designed to correspond to the form in which experts can describe their strategic knowledge: justifications for specific actions in specific situations. As will be explained in Section 2.3,ASK elicits justifications for choices among actions in terms of features of strategic situations and actions. ASK's design ensures that the features mentioned in justifications are operational; the features are well-defined functions and relations that hold over objects in a knowledge base representing the current state of problem solving.

Like all tools that elicit knowledge in executable form, ASK is based on the assumption of representational adequacy discussed above. There are two ways this assumption can fail: the computational model is inadequate for describing the desired strategy, or the set of terms in the existing knowledge representation is incomplete. The former problem is a function of the architecture, as discussed above. The problem of incomplete terms can be handled in an interactive tool if the user is given the chance to define new terms with the representational primitives provided by the architecture.

Since defining terms for a knowledge system is an operationalization task, it is a challenge to provide automated assistance. A promising approach is exemplified by PROTÉGÉ, a tool that helps the knowledge engineer define domain-specific instantiations of architecture-level representational primitives (Musen, 1989). PROTÉGÉ generates OPAL-class elicitation tools meant for the domain expert in which the vocabulary is fixed. ASK provides a means for defining new features in the context of eliciting justifications, as demonstrated in Section 2.4.6. By design ASK integrates the acquisition of new features and the acquisition of knowledge that uses the features.

2.2.3 Integrating mechanical generalization with interactive knowledge elicitation

Machine learning techniques are an obvious answer to the generalization aspect of representation mismatch. There are many well-established techniques for generalization from examples (Dietterich & Michalski, 1983). Because inductive

generalization is inherently underconstrained, these techniques all depend on some kind of *bias* to direct the learner toward useful or relevant generalizations (Mitchell, 1982; Utgoff, 1986). Bias can be provided to a learner by supplying a highly constrained generalization space, defined by the language for representing learned concepts, such as LEX's pattern-matching language (Mitchell, Utgoff, & Banerji, 1983). Bias can also come from the choice of features in the training examples, as in the feature vectors used by decision tree algorithms (Quinlan, 1986).

A knowledge acquisition tool can capitalize on existing generalization techniques if they are augmented with the appropriate bias. One approach would be to build the necessary bias into the tool. If the bias is itself important domain knowledge, however, this approach limits the usefulness of automating the knowledge acquisition process, since the tool would have to be modified for each domain. Instead, a knowledge acquisition tool can provide means for the *user* to contribute bias — to guide the generalization toward useful concepts. The user can contribute bias by carefully selecting training examples (Winston, 1985), by identifying their relevant features, and by evaluating machine-generated generalizations. While the human provides pedagogical input and evaluation of results, the tool can apply syntactic generalization operators and check for consistency with a database of training cases. The resulting human-machine synergy is a more powerful acquisition technique than either manual knowledge engineering or traditional inductive learning.

Knowledge-based learning techniques such as explanation-based learning (DeJong & Mooney, 1986; Mitchell, Keller, & Kedar-Cabelli, 1986) are strongly biased by the domain theory provided by the system builder. Inserting a human in the learning loop can help overcome the dependence of the learning technique on the quality of the built-in knowledge. For example, in an experiment with SOAR in the domain of algebraic simplification, a human intercedes during problem solving to help the system learn search-control knowledge (Golding, Rosenbloom, & Laird, 1987). When the system needs to choose among algebraic simplification operators for a specific equation, the human recommends an operator to apply or provides a simpler equation to solve. The system uses a domain theory of algebraic simplification to find useful chunks that generalize the situation (the class of equations) in which the recommended operator should be applied. In the absence of a complete domain theory, one can imagine the human pointing out relevant parts of the equation to chunk.

To integrate generalization techniques into a knowledge acquisition tool, the knowledge to be acquired must be represented in such a way that syntactic generalizations of statements in the representation correspond to semantic generalizations in the knowledge (see Lenat & Brown, 1984). For strategic knowledge, this means formulating the selection of actions in terms of classification. For example, a common technique for programs that learn search-control knowledge is to formulate the knowledge for selecting actions as pattern-matching expressions that identify situations in which operators would be usefully applied (Benjamin, 1987; Laird, Newell, & Rosenbloom, 1987; Minton & Carbonell, 1987; Mitchell, Utgoff, & Banerji, 1983; Silver, 1986). Because of this formulation, syntactic generalizations of the expressions to which an operator had been applied

during training correspond to classes of situations where the operator might be useful in the future.

ASK's representation of strategic knowledge is designed to exploit syntactic generalization operators. Knowledge about what action to do next is formulated as predicates that describe situations in which equivalence classes of actions are useful. In the absence of a theory to infer the utility of actions, ASK acquires strategic knowledge from people.

2.3 The ASK knowledge acquisition assistant

ASK is an interactive knowledge acquisition assistant. It acquires strategic knowledge from the user of a knowledge system, called the performance system. The strategic knowledge acquired by ASK is used by the performance system to decide what action to perform on each iteration of a control cycle. With additional strategic knowledge, the performance system should be able to make better decisions about what to do in various situations.

The basic approach is to elicit strategic knowledge from the user in the form of *justifications* for specific choices among actions, and then operationalize and generalize the justified choices in the form of *strategy rules* that associate situations with classes of appropriate actions.

This section presents an overview of the knowledge acquisition procedure, and then covers in more detail the strategy rule representation and the knowledge system architecture that supports it. Section 2.4 demonstrates ASK with examples from a knowledge system for planning workups of chest pain.

2.3.1 The knowledge acquisition dialog

ASK orchestrates a mixed-initiative dialog with the user. The basic steps in the knowledge acquisition dialog are shown in Figure 2.1.

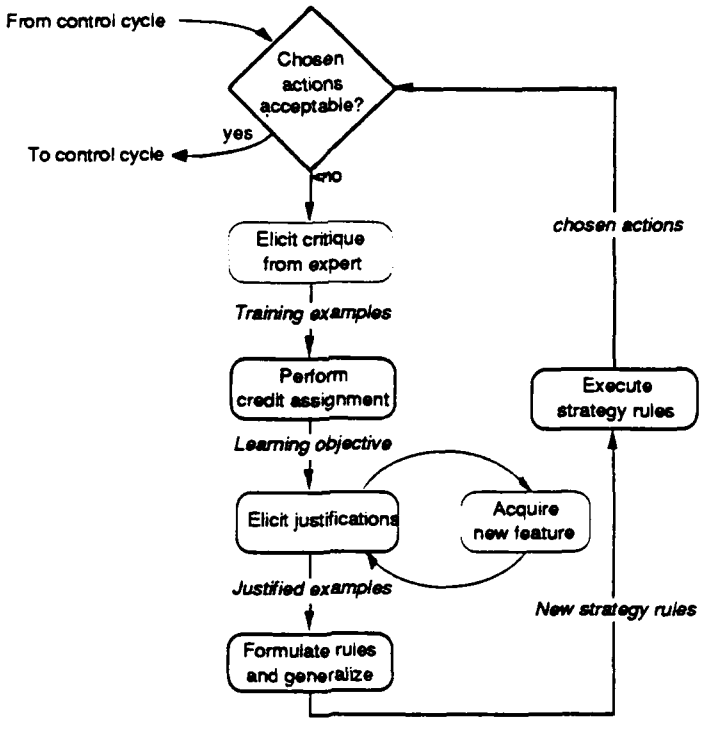


Figure 2.1. The ASK Knowledge Acquisition Dialog

ASK is invoked by the user of the performance system. At run time the performance system executes a simple control loop. On each iteration the system selects a set of recommended actions, the user picks one, and then the system executes it. The results of the action are recorded, and then the system continues by selecting the next set of recommended actions. If the user disagrees with the system's recommended actions on any iteration, she can interrupt the control loop and initiate a knowledge acquisition dialog.

The first step of the knowledge acquisition dialog is to elicit a critique from the user. A critique is a labeling of what the system did wrong in terms of choosing actions. The system recommends a set of actions at each iteration of the control cycle because they are all equally appropriate in the current situation, according to the existing strategic knowledge. The user critiques the system's choices by selecting an action that the system should have chosen (the positive example) and one that the system should not have chosen (the negative example). The positive and negative examples do not have to be in the set of the system's initial choices (which may be empty). The user also characterizes the the system's error in recommending actions, indicating, for instance, whether the positive example is merely preferred to the negative example or whether the negative example should not have been considered at all.

Next, ASK performs credit assignment analysis by examining how the current set of strategy rules matched the positive and negative examples. The output of this analysis is a learning objective that specifies what a new strategy rule would have to match and not match and what it should recommend in order to accommodate the user's critique and be consistent with existing strategy rules.

Then ASK elicits justifications from the user. From the user's perspective, justifications are explanations or reasons why an action should or should be recommended, in terms of "relevant features" of the current situation. From ASK's perspective, justifications are facts about the state of knowledge base objects in the current working memory of the performance system; the set of justifications corresponds to the set of features that should be mentioned in matching strategy rules. ASK suggests an initial "seed" set of justifications, based on how existing strategy rules fired. The user adds justifications by clicking on features of objects displayed in windows on the screen. The justification interface allows the user to browse through the knowledge base for relevant objects. If the set of existing features is inadequate, the user can define new features within the justification interface.

When the user indicates that she is finished and has specified a set of justifications that are sufficient to distinguish the positive and negative examples, ASK generates a new strategy rule from the justifications. The new strategy rule is generalized by syntactic induction operators to apply to a range of situations and an equivalence class of actions. For example, where a specific action appears in a justification, ASK puts a variable in the corresponding clause of a strategy rule. Similarly, if a justification mentions a specific value for a feature, ASK may build a strategy rule clause that matches a *range* of values for that feature.

Finally the new rule is paraphrased and the operational effects of the new rule are presented to the user for approval. If the user agrees that the new rule improves the system's choices of actions, the rule is added to the strategic knowledge base of the performance system, and the control cycle is continued.

Details of the knowledge acquisition dialog are demonstrated with examples in Section 2.4. First some background on the performance system architecture and the representation for strategic knowledge is required.

2.3.2 The MU architecture

ASK is integrated with an architecture for knowledge systems called MU (Cohen, Greenberg, & Delisio, 1987; Gruber & Cohen, 1987). As depicted in Figure 2.2, a performance system built in MU consists of a substantive knowledge base, typically for heuristic classification, and a strategic knowledge base for controlling actions. This division of knowledge is typical of architectures that support control knowledge, such as BB1 (Hayes-Roth, 1985). MU organizes the substantive knowledge as a symbolic inference network, where inferences are propagated from evidence to hypotheses by local combination functions. The inference network serves as the working memory of the system at runtime. The state of the network is abstracted by *control features*, which are functions, attributes, and relations over knowledge base objects.[1]z The strategic knowledge is organized in a separate component, which examines the state of working memory via control features and selects actions to execute. MU was designed to support a variety of experiments in strategic reasoning, so the architecture does not include a built-in problem solving

[1]Control features correspond to the *metarelations* in Clancey's tasks-and-metarules representation (Clancey & Bock, 1988).

method or control strategy. The strategy rule representation was developed for the study of knowledge acquisition in ASK.

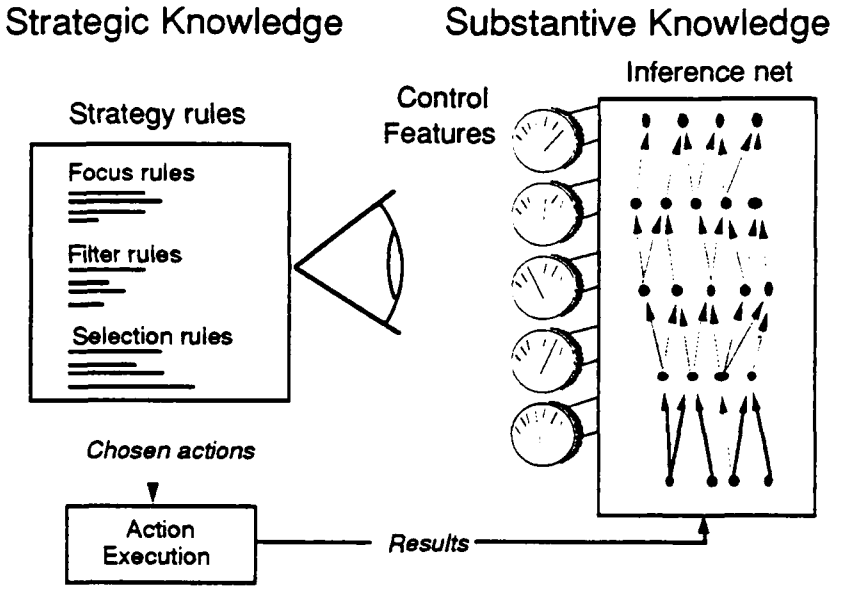


Figure 2.2. The MU architecture with strategy rules for control

2.3.3 Strategy rules

Strategic knowledge acquired by ASK is represented in the form of *strategy rules*, inspired by the metarules that represent diagnostic strategy in NEOMYCIN and HERACLES (Clancey, 1988; Clancey & Bock, 1988). Strategy rules map *strategic situations* to sets of recommended actions. Strategic situations are states of the working memory of a performance system. In the MU architecture, strategic situations are states of the inference network.

The strategy rule control cycle, shown in Figure 2.3, specifies how strategy rules are applied in a performance system to decide among actions. At each iteration of the control cycle, strategy rules recommend the actions that are appropriate to perform next. There are three types of recommendations, corresponding to three categories of strategy rules. *Focus rules* propose a set of possible actions at each iteration. *Filter rules* prune actions that violate constraints. *Selection rules* pick out subsets of the proposed and unpruned actions that are most desirable in the current situation to form the final set of recommended actions. One of the actions in the recommended set is chosen by the user and executed. The effects of executing the action are then propagated through working memory.

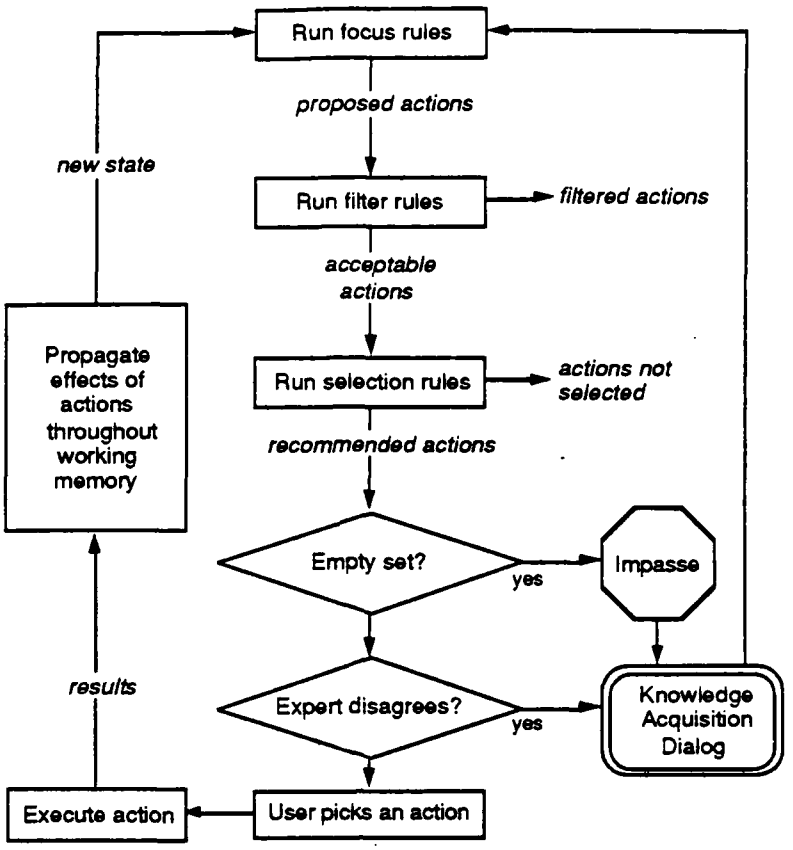


Figure 2.3. The Strategy Rule Control Cycle

The strategy rule control cycle corresponds to the "method" of task-level architectures described in Section 2.2.1. It specifies how strategic knowledge is brought to bear in the decision about what action to do next. The propose-filter-select algorithm defines three roles for strategic knowledge: specifying the conditions under which actions might be applicable, inappropriate, and preferable. Its design stipulates that actions are chosen iteratively, waiting for the effects of the execution of the previous action before making the current decision. The algorithm also assumes that context of the decision, the strategic situation, is defined in terms of currently available features of the state of the performance system. Thus, strategy rules are *not* general-purpose control rules, useful for writing arbitrary programs. Rather, the strategy rule control cycle supports a style of reasoning that has been called *reactive planning* (Agre & Chapman, 1987; Chapman & Agre, 1987; Firby, 1987; Kaelbling, 1987). The form of strategic knowledge is restricted to facilitate automated knowledge acquisition. The consequences of this design are made explicit in later sections.

The left-hand side (If part) of a strategy rule is a conjunctive expression, with variables, that specifies a strategic situation and the set of recommended actions for that situation. The left-hand side expression matches against the values of control features that reflect the properties and dynamic state of objects in working memory, including objects that represent actions. The right-hand side (Then part) of a strategy rule indicates whether the matching actions should be proposed, filtered, or selected in the matching situation.

2.3.4 Examples from the chest pain domain

Here are some examples of strategy rules and control features from a system for planning workups for chest pain that will be used to demonstrate ASK in Section 2.4.

The following *focus rule* proposes actions that are general questions (e.g., age, sex, etc.) when the the set of active hypotheses, called the differential, is empty.

```
Rule  Ask-intake-questions     a focus rule
  "Ask general questions when at a loss."
If: (IS (differential) :EMPTY)
    (IN ?ACTION (members-of general-questions))
Then: (PROPOSE ?action history-and-exam)
```

The strategic situation in this rule is specified by the condition that the value of the `differential` object is empty. The set of recommended actions is generated by the relation `members-of` applied to the object `general-questions`, which is a class of actions. The right-hand side operator `PROPOSE` specifies that the values bound to the variable `?ACTION` should be proposed under these conditions, and that the goal `history-and-exam` should be posted.

The expression `(differential)` refers to the set of hypotheses on the differential. It is a control feature defined in the MU inference network as:

```
VALUE  of  DIFFERENTIAL          a control feature
  "The set of active hypotheses"
SET-OF ?Hypothesis IN hypotheses SUCH-THAT
  trigger-level OF ?Hypothesis IS triggered AND
  level-of-support OF ?Hypothesis IS-NOT disconfirmed
 OR
  level-of-support OF ?Hypothesis IS-AT-LEAST supported
```

Another focus rule, shown below, is complementary to `Ask-intake-questions`. It proposes actions that potentially provide diagnostic evidence when the differential is *not* empty, and labels this state with the goal `gather-evidence-for-differential`.

```
Propose-diagnostic-evidence        a focus rule
  "Gather evidence for current hypotheses."
If: (IS (differential) :NONEMPTY)
    (IN ?ACTION (potential-evidence differential))
Then: (PROPOSE ?ACTION gather-evidence-for-differential)
```

`(potential-evidence differential)` is a control feature that returns the set of actions that are potentially diagnostic for hypotheses on the differential. This set is computed dynamically by a function that calls a MU service for analyzing the inference network (Cohen, Greenberg, & Delisio, 1987).

A very simple *filter rule* prevents actions from being recommended if they have already been executed. In some domains actions may be executed repeatedly. That is why the don't-repeat policy is encoded in the following rule instead of built in to the basic control loop.

```
Filter-executed-actions                    a filter rule
  "Do not repeat actions"
If: (IS (executed? ?ACTION) yes)
Then: (FILTER ?ACTION)
```

The following *selection rule* is enabled under the goal history-and-exam. It recommends those actions that are cheap to perform and that can potentially produce data that would trigger new hypotheses.

```
Select-cheap-triggering-data                a selection rule
  "Prefer cheap actions that might trigger hypotheses."
If: (IN history-and-exam (current-goals))
    (IS (potentially-triggered-by ?ACTION) :NONEMPTY)
    (≤ (cost ?ACTION) cheap)
Then: (SELECT ?ACTION)
Shadows: Select-triggering-data, select-free-evidence,
         select-cheap-evidence
```

The terms current-goals, potentially-triggered-by, and cost refer to control features. The set of actions recommended by this rule are those with some hypotheses on their potentially-triggered-by feature and whose cost feature is not more than cheap. The feature potentially-triggered-by is computed from the definitions of triggering conditions for hypotheses, stated in a rule-like form. For example, the hypothesis classic-angina is triggered when "the chief-complaint is pain or pressure and pain-quality is vise-like and the chief-complaint-location is substernal." This rule will recommend the action of asking for the chief-complaint-location because it potentially triggers a hypothesis and it is cheap.

2.3.5 The shadowing relation among strategy rules

Within each strategy rule category (focus, filter, selection), rules are matched in an order specified by a precedence relation called *shadows*, which is a partial order based on the generality of left-hand sides. If a rule succeeds (matches some objects), then the more general rules that it shadows are pruned (prevented from being fired). Generality is defined in terms of the features mentioned in a rule and the range of values specified for each feature. For example, the selection rule shown above, Select-cheap-triggering-data, shadows (takes precedence over) more general rules mentioning the same features. It shadows the more general rule Select-cheap-evidence, which recommends any action that is cheap. In turn, Select-cheap-evidence shadows the rule Select-cheap-evidence because the former matches actions with costs of cheap *or* free. The global effect of a family of selection rules in which the more specific rules shadow the more general is to choose those actions judged to be acceptable by the most constraining criteria. The shadows relation is a symbolic alternative to a numeric function for combining the recommendations of each rule into a single measure of utility. Further details can be found in (Gruber, 1989).

2.4 A knowledge acquisition dialog with ASK

In this section, ASK will be demonstrated in the context of a performance system that generates diagnostic workups for patients reporting chest and abdominal

pain. The performance system is a reimplementation of the MUM knowledge system (Cohen et al., 1987). MUM's task is called *prospective diagnosis*, which is to choose diagnostic actions as a physician would, asking questions in an intelligent order and balancing the potential costs of diagnostic tests and trial therapy with the evidential and therapeutic benefits.

2.4.1 What the performance system already knows

In experiments with ASK, the performance system is given MUM's substantive knowledge about the diagnosis of chest pain, implemented in the MU architecture in an inference network. The inference network contains hypotheses, data-gathering actions, intermediate conclusions, and combination functions that represent inferential relations such as the evidential support for hypotheses given patient data. MUM's original strategy was written by knowledge engineers in Lisp. In the ASK experiments, the strategic knowledge is represented in strategy rules.

In the dialog shown here, the performance system starts with a small but incomplete set of strategy rules, and the user extends them to improve strategic performance. ASK can be used without any existing strategy rules. In an separate experiment reported in (Gruber, 1989), ASK was used to acquire a set of strategy rules that replicates the original MUM strategy. However, since ASK makes use of existing strategy rules and control features in acquiring new strategic knowledge, it can be more helpful in specializing an existing strategy than in building a strategy from scratch. Thus the dialog in this section will show ASK being used to extend an existing set of rules that represent a general strategy for prospective diagnosis.

2.4.2 Running the performance system

A MU performance system runs the basic control loop that was introduced in Section 2.3.3. At each iteration, strategy rules recommend some set of actions as candidates. From the system's point of view, these recommended actions are equivalent. Given the current strategic knowledge, the system could select among them arbitrarily. The user of a MU system is given the choice to "break the tie" and pick one action to execute. In the chest pain application, executing an action typically causes a request for data (e.g., symptoms or test results). That data is entered into the inference network, where it may change the evidential support for active hypotheses and trigger new hypotheses.

We begin the knowledge acquisition dialog at a point at which the user has already run the performance system through the first several actions in a case (namely, the cheap and easy questions about the history and the physical examination data). At this point, the system has run out of cheap actions and the `Propose-diagnostic-evidence` rule (Section 2.3.4) recommends a set of diagnostic actions. The user has the option to pick one of the recommended actions for execution or to teach the system to refine its strategy.

The following menu shows the system offering a set of recommended actions during an iteration of the control cycle of the performance system. Instead of choosing an action, the user initiates the dialog with ASK to "teach the system to improve its choices." (An item with a box drawn around it signifies that the user

has selected it with the mouse.) The user set up this diagnostic situation because it demonstrates a weakness in the system's strategy. The system needs to be more selective in choosing among diagnostic tests and trial therapeutic actions such as the seven offered in the menu.

Please choose something to ask or perform.

BARIUM-SWALLOW
EKG
GASTROSCOPY-WITH-BIOPSY
NITROGLYCERINE-TX
STRESS-TEST
UPPER-GI-SERIES
VASODILATOR-TX

Teach the system to improve its choices.
Explain why these actions were chosen.
Help

2.4.3 Eliciting the user's critique

ASK elicits a critique from the user by presenting the list of the system's chosen actions and asking what should have been done differently. It first asks for the general category of error, to help determine whether the problem is with focus, filter, or selection rules:

Please explain why you disagree with the system's choices.

BARIUM-SWALLOW
EKG
GASTROSCOPY-WITH-BIOPSY
NITROGLYCERINE-TX
STRESS-TEST
UPPER-GI-SERIES
VASODILATOR-TX

One or more of these actions are PREFERRED to the others.
One or more of these actions should NOT have been suggested.
Some action NOT MENTIONED HERE should have been suggested.
Help

Then it asks for a positive example, an action that should have been recommended, and a negative example, an action that should not have been recommended. It is assumed that the user will choose a positive example that is representative of a class of actions that should be recommended in this situation, and a negative example that represents a class of actions to distinguish in this situation. In the interaction shown below the user indicates that the action EKG should have been distinguished from the action Upper-GI-series, which is a reasonable alternative (i.e., a near miss).

Which action would you have chosen?

BARIUM-SWALLOW
[EKG]
GASTROSCOPY-WITH-BIOPSY
NITROGLYCERINE-TX
STRESS-TEST
UPPER-GI-SERIES
VASODILATOR-TX

an action not shown here
Help

Which of the system-selected actions would you NOT have chosen?

BARIUM-SWALLOW
GASTROSCOPY-WITH-BIOPSY
NITROGLYCERINE-TX
STRESS-TEST
[UPPER-GI-SERIES]
VASODILATOR-TX

They are all as appropriate as STRESS-TEST.
Help

2.4.4 Credit assignment analysis

Using the information provided by the user, ASK performs a credit assignment analysis. The credit assignment algorithm examines how existing strategy rules matched in this situation and determines the requirements for a new rule that would account for the critique. The algorithm makes strong use of the distinction between focus, filter, and selection rules and the way they are applied in the strategy rule control cycle. For example, if the positive example was not proposed by any focus rules, the algorithm prescribes learning a focus rule that proposes it. Alternatively, if both the positive and negative examples are recommended by selection rules, then the algorithm prescribes learning a selection rule that matches the positive example, fails to match the negative example, and shadows the selection rules that recommended the negative example. In the sample session, ASK determines that it needs to acquire a selection rule, specializing the `Propose-diagnostic-evidence` rule, such that the new rule matches EKG and does not match Upper-GI-series. The complete credit assignment algorithm can be found in (Gruber, 1989).

2.4.5 Eliciting justifications

In the next stage of the dialog, the user provides justifications for choosing the positive example over the negative example. Justifications are specified as features of the current strategic situation and features of actions. In the example session, the strategic situation is characterized by the state of hypotheses on the differential. A feature shared by the actions recommended by the system (including the positive and negative examples) is that they potentially provide evidence for hypotheses on the differential. In current example, the user must provide additional justifications that distinguish the positive example EKG from the negative example Upper-GI-series.

The user interface for asserting justifications consists of two windows containing mouse-sensitive text. The "relevant objects window" displays the values of features of a set of objects from the knowledge base. The "justifications window" contains a list of justifications in the form of natural language sentences. Each justification is a description of the value of a feature of some relevant object.

ASK initializes the relevant objects window with a set of knowledge base objects that might be relevant to the current control decision. An object is considered relevant if it is one of the positive or negative examples (actions), a current goal, an instance of a class representing some aspect of the global state of the inference network, or if it is mentioned in a strategy rule matching the positive or negative examples. The user is provided with tools for browsing the knowledge base to find additional relevant objects.

ASK also initializes the list of statements in the justification window with *seed justifications* which represent the system's reasons for selecting the current actions. Seed justifications are derived from the clauses of strategy rules matching the positive and negative examples. In the windows shown below objects and justifications have been seeded by ASK.

Objects Relevant to the Control Decision

CRITICAL-HYPOTHESES
Value: classic-angina, unstable-angina
CURRENT-GOALS
Value: gather-evidence-for-differential
DIFFERENTIAL
Potential-evidence: barium-swallow, ekg, gastroscopy-with-biopsy, nitroglycerine-tx,
Potentially-conclusive-evidence: barium-swallow, ekg, gastroscopy-with-biopsy, str
Value: classic-angina, esophagitis, esophageal-reflux, pericarditis, unstable-angina,
EKG
Applicability: APPLICABLE
Classes: diagnostic-tests.
Cost: LOW
Executed?: NO
Potentially-confirms: classic-angina, prinzmetal-angina, unstable-angina, variant-an
more below

Justifications for the Current Control Decision

GATHER-EVIDENCE-FOR-DIFFERENTIAL is in the CURRENT-GOALS.
EKG is in the POTENTIAL-EVIDENCE of DIFFERENTIAL.
UPPER-GI-SERIES is in the POTENTIAL-EVIDENCE of DIFFERENTIAL.

The user asserts a justification by selecting a feature of one of the objects presented in the relevant objects window. When a justification is selected, ASK paraphrases the fact in the justifications window. In the following interaction, the user indicates that EKG should have been chosen because it has low cost. Using the mouse, the user selects the statement "Cost: low" from the relevant objects window, and the statement "The COST of EKG is low" shows up in the justification window, as depicted below.

Objects Relevant to the Control Decision

CRITICAL-HYPOTHESES
Value: classic-angina, unstable-angina
CURRENT-GOALS
Value: gather-evidence-for-differential
DIFFERENTIAL
Potential-evidence: barium-swallow, ekg, gastroscopy-with-biopsy, nitroglycerine-tx,
Potentially-conclusive-evidence: barium-swallow, ekg, gastroscopy-with-biopsy, str
Value: classic-angina, esophagitis, esophageal-reflux, pericarditis, unstable-angina,
EKG
Applicability: APPLICABLE
Classes: diagnostic-tests.
Cost: LOW
Executed?: NO
Potentially-confirms: classic-angina, prinzmetal-angina, unstable-angina, variant-an
more below

Justifications for the Current Control Decision

GATHER-EVIDENCE-FOR-DIFFERENTIAL is in the CURRENT-GOALS.
EKG is in the POTENTIAL-EVIDENCE of DIFFERENTIAL.
UPPER-GI-SERIES is in the POTENTIAL-EVIDENCE of DIFFERENTIAL.
The COST of EKG is low.

At this point the user could tell ASK that she was finished. If the set of justifications satisfied the learning objective, ASK would then turn the justifications into a new strategy rule. In this session, however, the user wishes to add more justifications. In particular, the user wants to say that EKG is appropriate in this situation not only because it has low cost, but also because takes little time to perform. To be able to say this in the language of justifications, the user needs to define a new feature..

2.4.6 Acquiring a new feature

To define a new feature is to implement an attribute, function, or relation over some set of objects in the knowledge base. ASK can help the user define a new feature. Playing the role of a knowledge engineer, ASK elicits the information needed to implement the feature in the MU architecture. The interaction below shows the user defining a new feature called "time required." The user starts by clicking on the EKG object in the relevant objects window, bringing up the following menu:

EKG

Display unit
Remove Object
Apply an existing feature
Define a new feature

What shall we call the new feature of EKG?

TIME-REQUIRED

After obtaining a name for the feature, ASK needs to determine its general type. The type of a feature is a symbol-level property, dependent on the knowledge

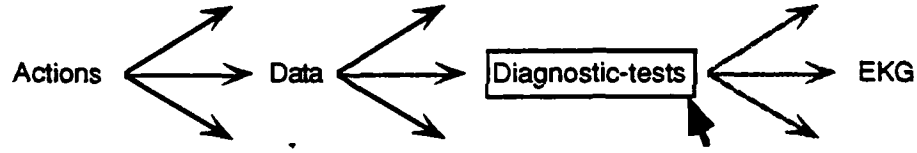
base architecture. MU supports several varieties of control features, many of which are best implemented by knowledge engineers (e.g., dynamic relations written in Lisp). ASK knows about how features are implemented in MU and makes it possible to acquire some of the more simple features, such as static attributes, interactively. To help make architecture-dependent terms such as "inferential value" concrete to the user, ASK offers instances of features types from the current knowledge base as exemplars. In the menu below, the user indicates that the time-required feature is an attribute of actions, analogous to the cost feature.

What kind of feature is TIME-REQUIRED?

an attribute of actions (like COST)
a class of actions (like DIAGNOSTIC-TESTS)
an object (like DIFFERENTIAL)
an inferential value computed by rules (like LEVEL-OF-SUPPORT)
a dynamic relation (like POTENTIALLY-CONFIRMS)
Help

To complete the definition of a static attribute, ASK elicits information about the domain, data type, possible values, order, cardinality, and default value for the feature, and constrains the user's choices whenever possible.

To which of these parent classes of EKG will Time-required apply?



What possible values might Time-required take?

Yes or No (like EXECUTED? of EKG)
one of a list of words (like COST of EKG)
a member of a KB class (like CURRENT-GOALS)
a number (like VALUE of AGE)
a duration of time (like VALUE of EPISODE-DURATION)
Help

Please enter the possible values of Time-Required.

(e.g., values for COST are: free, cheap, low, moderate, medium-high high not-insured)

IMMEDIATE FEW-MINUTES AN-HOUR FEW-HOURS A-DAY FEW-DAYS WEEKS MONTHS

Is there an ordering over the possible values for Time-required?

Yes
No
Help

Can there be more than one Time-required?

Yes
No
Help

Please choose a default value for Time-required.

immediate
few-minutes
an-hour
few-hours
a-day
few-days
weeks
months
No default is applicable
Help

Once the intensional properties of the feature are acquired, the values of the feature applied to the elements of its domain are elicited. For static attributes, ASK presents a table of the objects to which it applies, and the user specifies the value of the feature for each object. In the current example, the user enters the value of the time-required feature for all diagnostic tests, including the training examples EKG and Upper-GI-series. The table below shows the value of time-required for EKG, after it was entered by the user.

Time-required of Diagnostic-tests

Angiogram	unknown
Barium-swallow	unknown
Cardiac-enzyme	unknown
Chest-xray	few-hours
Cholesterol-level	unknown
Echo-cardiogram	unknown
EKG	few-minutes
Flat-plate-of-the-abdomen	unknown
Gall-bladder-series	unknown

More below

2.4.7 Using the new feature in justifications

When the expert has finished defining time-required, the system can use it as any other feature and ASK can offer it as a possible justification. The dialog now returns to the justification interface, where the user selects the time-required as a justification for choosing EKG over Upper-GI-series:

Objects Relevant to the Control Decision

CRITICAL-HYPOTHESES
    Value: classic-angina, unstable-angina
CURRENT-GOALS
    Value: gather-evidence-for-differential
DIFFERENTIAL
    Potential-evidence: barium-swallow, ekg, gastroscopy-with-biopsy, nitroglycerine-tx,
    Potentially-conclusive-evidence: barium-swallow, ekg, gastroscopy-with-biopsy, str
    Value: classic-angina, esophagitis, esophageal-reflux, pericarditis, unstable-angina,
EKG
    Applicability: APPLICABLE
    Classes: diagnostic-tests.
    Cost: LOW
    Executed?: NO
    Potentially-confirms: classic-angina, prinzmetal-angina, unstable-angina, variant-an
    Potentially-triggered: None.
    Time-required: FEW-MINUTES
    Value: unknown

more below

Justifications for the Current Control Decision

GATHER-EVIDENCE-FOR-DIFFERENTIAL is in the CURRENT-GOALS.
EKG is in the POTENTIAL-EVIDENCE of DIFFERENTIAL.
UPPER-GI-SERIES is in the POTENTIAL-EVIDENCE of DIFFERENTIAL.
The COST of EKG is low.
The TIME-REQUIRED of EKG is few-minutes.
The TIME-REQUIRED of UPPER-GI-SERIES is a-day.

At this point in the dialog, the user has indicated that the cost and time-required of actions are factors to consider when choosing actions. The first three justifications represent the factors that the system would consider and were suggested by ASK. The user could have removed some of the seed justifications but did not in this case. From the combined set of justifications ASK can generate a new strategy rule.

2.4.8 Generating and generalizing a strategy rule

Given the user's justifications, ASK formulates a new strategy rule that accounts for the expert's critique of the system's performance. The new rule causes the user's preferred action to be selected on the next iteration.

The left-hand side of the new rule is constructed by transforming the list of justifications into left-hand side clauses. The transformation from justifications to rule clauses is fairly straightforward. The internal representation of justifications is very similar to the clause form of strategy rules. The right-hand side recommendation (in this case, SELECT) was decided by the credit assignment analysis. In the current example, ASK forms the following rule:

```
IF  (IN gather-evidence-for-differential (current-goals))
    (IN ?ACTION (potential-evidence differential))
    (≤ (cost ?ACTION) low)
    (= (time-required ?ACTION) few-minutes)
THEN (SELECT ?ACTION)
```

In the process of forming rule clauses from justifications, ASK applies generalization operators. One operator is called *turning constants to variables.* In the strategy rule above, references to EKG have been replaced with the free variable ?ACTION, which is bound at runtime by the strategy rule interpreter to each action that has been proposed and has not been filtered. The result is that the rule recommends the class of actions sharing the features of EKG in the justifications: the cost and time required.

Another generalization operator is *extending the reference* of a feature from an test of equality to a test over some range or set of permissible values. In the example strategy rule, the ≤ operator specifies that the third clause will succeed when the action has any value of cost equal to or less than low. ASK used a heuristic for applying this generalization; it found another selection rule that used ≤ for the cost feature.

In this example, however, ASK has no a priori information to help in extending the reference of the new feature, time-required. It asks the user for guidance by posing hypothetical variants on the current case to obtain boundary conditions on the acceptable range for the time-required clause in this rule. Since ASK lacks common sense, it has to ask whether the user would still accept the EKG if it takes no time at all:

What if the Time-required of EKG were IMMEDIATE?
Would you still choose it?

[Yes]
No
Help

Then ASK offers near-miss cases:

What if the Time-required of EKG were AN-HOUR?
Would you still choose it?

[Yes]
No
Help

What if the Time-required of EKG were FEW-HOURS?
Would you still choose it?

[Yes]
No
Help

What if the Time-required of EKG were A-DAY?
Would you still choose it?

Yes
[No]
Help

Given this information, ASK replaces the clause

```
(= (time-required ?ACTION) few-minutes)
```

with the clause

```
(≤ (time-required ?ACTION) few-hours)
```

2.4.9 Verifying a rule

To evaluate the face validity of the generated rule, ASK presents a paraphrased translation to the user for verification. It also shows the operational consequences of the rule.

Knowledge Acquisition Dialog

I would paraphrase your advice as:
Select an action when
a current goal is gather-evidence-for-differential, and
the action is in the potential-evidence of differential, and
the cost of the action is less than or equal to low, and
the time-required of the action less than or equal to few-hours.

Considering this advice, the system would choose this action only:
EKG

Is this an improvement?

Please verify this advice. It is based on your justifications.

I agree with this rule.
I would like to change the justifications.
Let me look at it again.
Help

This completes one session of the knowledge acquisition dialog. With the new strategy rule, the performance system now recommends only the positive example, EKG, when the goal is to gather evidence and the actions are potentially diagnostic. The new selection rule fails to match the negative example and the other proposed actions, and it shadows the more general rule that formerly matched all seven actions.

The next subsection demonstrates how ASK can be used to acquire tradeoffs in a utility space. It is not essential to understanding the basic approach.

2.4.10 Acquiring tradeoffs

The strategy rule just acquired is one of a family of rules that together constitute a strategy for selecting diagnostic actions. Selection rules can be viewed as tradeoffs among features, and a family of selection rules represents a set of acceptable tradeoffs. The new rule specifies that a moderate amount of time is acceptable if the cost is low and the diagnosticity is moderate.

In terms of utility theory, the new rule occupies a region in a space with dimensions defined by the features measuring diagnosticity, cost, and timeliness. Points in this space can be interpreted as the values of a multiattribute utility function (Keeney & Raiffa, 1976). The dimensions are attributes and the regions

represent values of equivalent utility. The shadows relation among rules corresponds to a partial order over values of utility; some regions have higher utility than others in the same attribute space. For example, because of the shadows relation, the new rule takes precedence over selection rules that only mention cost or time-required. The region corresponding to the new rule can be interpreted as having higher utility. In other words, actions selected by the new rule are preferred over actions that would have been selected by shadowed rules.

To illustrate how ASK can be used to acquire other tradeoffs in the same space, this subsection sketches a second session where the user finds an exception to an existing rule.

In this second scenario, the user runs the performance system on a case where initial data provided evidence that the patient could have a very serious condition which requires immediate diagnosis. In this situation, the system suggests a set of actions that are potential evidence for hypotheses on the differential and have low cost. However, the user indicates that the system should ignore cost and concentrate on evidence that is potentially *conclusive* for hypotheses that are *critical*. The relevant objects and justification windows appear as follows:

Objects Relevant to the Control Decision

CRITICAL-HYPOTHESES
    Value: classic-angina, unstable-angina.
CURRENT-GOALS
    Value: gather-evidence-for-differential.
DIFFERENTIAL
    Potential-evidence: gastroscopy-with-biopsy, nitroglycerin-tx, stress-test, upper-gi-seri
    Potentially-conclusive-evidence: gastroscopy-with-biopsy, stress-test, upper-gi-series
    Value: classic-angina, esophagitis, esophageal-reflux, esophageal-spasm, unstable-a
STRESS-TEST
    Applicability: applicable
    Classes: diagnostic-tests.
    Cost: medium

more below

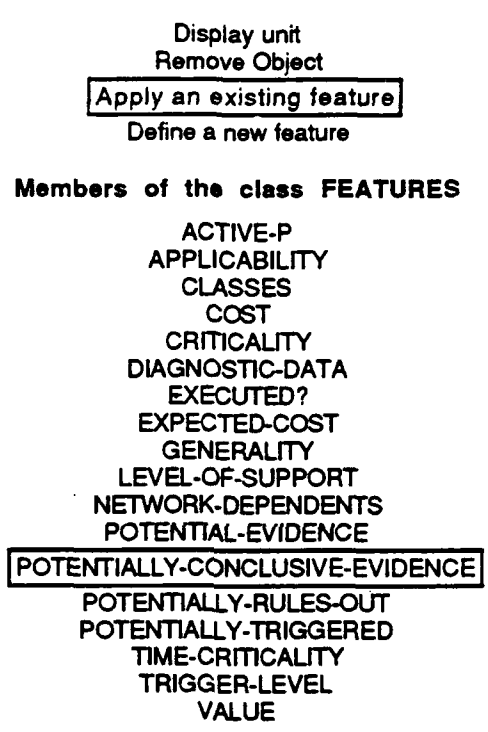
Justifications for the Current Control Decision

GATHER-EVIDENCE-FOR-DIFFERENTIAL is in the CURRENT-GOALS.
NITROGLYCERINE-TX is in the POTENTIAL-EVIDENCE of DIFFERENTIAL.
STRESS-TEST is in the POTENTIAL-EVIDENCE of DIFFERENTIAL.
The COST of NITROGLYCERINE-TX is low.
The COST of STRESS-TEST is medium.

The positive example is Stress-test, which was not selected by the system because its cost was more than low. The negative example is Nitroglycerine-tx, which was selected by the system. The justifications in the window shown above were seeded by ASK; they correspond to the clauses of the strategy rules that picked Nitroglycerine-tx and not Stress-test.

In the justification session, the user tells ASK to consider *conclusive* evidence for *critical* hypotheses. The set of critical hypotheses is already represented by a

knowledge base object. Critical-hypotheses is defined as a set of hypotheses that are active (and therefore on the differential) and time-critical (a feature of hypotheses). The relationship between conclusive evidence and critical hypotheses is *not* currently represented by a feature. The relationship *is* currently defined for the set of hypotheses on the differential. Since the set of critical hypotheses and the differential share the same domain, the feature implementing the potentially-conclusive-evidence relationship can be applied to the critical-hypotheses object. The user accomplishes by clicking on the critical-hypothesis object and performing the operations shown in the following windows.



The feature potentially-conclusive-evidence was conveniently defined to work for any set of hypotheses, and critical-hypotheses is a set of hypotheses. As a result, when the user applies the feature to critical-hypotheses, the set of potentially-conclusive evidence for critical hypotheses is immediately computed. The newly-applied feature is displayed in the relevant objects window, and becomes available as a justification. The updated relevant objects window shows the value of the feature as the singleton set containing the action Stress-test. In the window shown below the user selects this fact as a justification for choosing the stress test.

Objects Relevant to the Control Decision

CRITICAL-HYPOTHESES
Potentially-conclusive-evidence: STRESS-TEST.
Value: classic-angina, unstable-angina.
CURRENT-GOALS
Value: gather-evidence-for-differential.
DIFFERENTIAL
Potential-evidence: gastroscopy-with-biopsy, nitroglycerin-tx, stress-test, upper-gi-seri
Potentially-conclusive-evidence: gastroscopy-with-biopsy, stress-test, upper-gi-series
Value: classic-angina, esophagitis, esophageal-reflux, esophageal-spasm, unstable-a
STRESS-TEST
Applicability: applicable
Classes: diagnostic-tests.
Cost: medium

more below

Justifications for the Current Control Decision

GATHER-EVIDENCE-FOR-DIFFERENTIAL is in the CURRENT-GOALS.
NITROGLYCERINE-TX is in the POTENTIAL-EVIDENCE of DIFFERENTIAL.
STRESS-TEST is in the POTENTIAL-EVIDENCE of DIFFERENTIAL.
The COST of NITROGLYCERINE-TX is low.
The COST of STRESS-TEST is medium.
STRESS-TEST is in the POTENTIALLY-CONCLUSIVE-EVIDENCE of CRITICAL-HYPOTHESES.

With this set of justifications, ASK generates the rule paraphrased to the user as follows:

Select an action when
a current goal is gather-evidence-for-differential, and
the action is in the potential-evidence of differential, and
the action is in the potentially-confirming-evidence of critical-hypotheses, and
the cost of the action is ignored.

The final clause of the rule is a positive form of the *dropping conditions* generalization operator. It specifies explicitly that the cost criterion, which was mentioned in the system's existing rule, should be overridden by this new rule. The ignore clauses are used in determining the shadowing relationship among strategy rules (Section 2.3.5). This new rule will shadow the existing rule. The operational effect is that the actions that are potentially conclusive for critical hypotheses will be selected regardless of cost, if there are any such actions and hypotheses; otherwise actions that provide evidence for any active hypotheses and have low cost will be selected.

2.5 Experience using ASK

This section reports briefly on some test sessions performed to evaluate ASK. More detailed analysis of these experiments and the positive and negative results may be found in (Gruber, 1989).

ASK has been tested for the *prospective diagnosis* task (Cohen, Greenberg, & Delisio, 1987) in the domain of chest pain, which is the problem addressed by the MUM system and used as an example performance system in this article. The original MUM strategy, the strategic phase planner described in (Cohen et al., 1987), was written by a knowledge engineer as a set of knowledge sources implemented in Lisp. ASK was used by its designer to (re)acquire MUM's strategic knowledge from scratch in the form of strategy rules.

ASK was also tested with the physician who served as the domain expert for MUM. He was able to add domain-specific strategic knowledge to an existing general strategy in dialogs like those demonstrated in Section 2.4. In one session, the original domain expert taught a colleague how to use ASK. In general, this experience suggested that the following conditions are important for success at helping the domain expert teach a diagnostic strategy to ASK:

- The relevant control features are defined in advance (e.g., the potentially-conclusive-evidence relation of Section 2.4.10) or are analogous to existing features (e.g., the definition of time-required, which is analogous to cost, can be elicited by example as shown in Section 2.4.6). When new features have no analog, then it may require knowledge engineering skills to define them. The problem of defining features is discussed in Section 2.6.3.
- The user understands the opportunistic control model that underlies the strategy rule representation. If the user does not understand how the strategic knowledge is used, he or she may not give ASK useful information upon which to build strategy rules. For example, when the second physician used ASK for the first time, he tried to get it to follow a procedure-like plan: e.g., ask *all* the history and examination questions before proposing *any* diagnostic tests. This caused ASK to construct an overly-general strategy rule, as described in Section 2.6.3.

Some representational limitations of the strategy rule approach to control were revealed in another experiment in which ASK was used to reimplement NEOMYCIN's diagnostic strategy. One difference between ASK's strategy rules and NEOMYCIN's tasks and metarules (Clancey, 1988; Clancey & Bock, 1988) is the way in which problem-solving state is represented. In NEOMYCIN, metarules are invoked by tasks, and tasks are invoked like subroutines with arguments. Some of the problem solving state is represented by the calling stack for task invocation. In addition, metarules access and set global variables. These computational properties make certain kinds of strategic knowledge easier to represent. The task structure serves as a natural representation for goal-directed control, and the global variables and task arguments encourage a strategy with a persistent focus on the "current hypothesis" and "current finding." In contrast, ASK's strategy rules have no hierarchical calling structure and cannot set global variables. As a consequence, it is difficult to implement a goal-directed (top-down) strategy or to manipulate the differential as a data structure. ASK's representation and the corresponding elicitation metaphor is more suited to acquiring an opportunistic strategy.

In principle, one can completely reproduce the observable behavior of the NEOMYCIN strategy using ASK, because the strategy rule language together with MU's control features are Turing complete. In practice, knowledge engineering

skills were required to coerce the desired behavior from strategy rules, mainly by defining control features. For example, the engineer using ASK had to define special control features to correspond to NEOMYCIN's "current hypothesis" and "pursued hypothesis" which were stated more naturally with metarules and variables in the NEOMYCIN language. The engineering effort went into defining sophisticated features. ASK is more helpful for building up associations between existing features and actions in strategic decisions.

2.6 Analysis: scope of applicability, assumptions, and limitations

Although the approach taken with ASK is independent of any domain, it necessarily sacrifices generality for power. The ASK approach commits to a method of applying strategic knowledge that iteratively chooses among individual actions, employs strategy rules for the representation, and bases new knowledge on justifications of choices of actions. As a consequence, ASK has limited scope and requires some strong assumptions. This section will characterize the scope of applicability of ASK in terms of properties of a class of performances tasks and will explicate the critical assumptions and limitations that are inherent in the approach.

2.6.1 Characteristics of tasks to which ASK applies

The problems to which ASK can be applied are those for which expert strategy is essential to the performance task and for which the strategy rule knowledge representation and MU architecture are adequate. This is not a circular definition; it states that the applicability of the acquisition tool depends largely on the adequacy of the performance representation. Representational adequacy is judged with respect to the class of performance tasks and the problem-solving method for which a representation is designed (see Section 2.2.1). So, ASK's applicability will be characterized in terms of tasks for which the strategy rule representation and the strategy rule control cycle are appropriate.

The major characteristics of tasks to which ASK would apply are as follows:

Actions can be selected one at a time (as opposed to sequences of action).

A task for which this characteristic often holds is reactive planning for robots, where uncertainty about the world and real-time constraints necessitate acting without projection. Robots controlled by reactive planners select actions on the basis of immediate features of the environment, without projecting the consequences of several possible sequences of actions and picking the best sequence.

A task for which selecting actions one at a time is *not* appropriate is planning a set of drugs to cover an infection. MYCIN's therapy algorithm, for example, selects a collection of drugs to cover a set of infectious organisms using an algorithm written in Lisp (Clancey, 1984). This task necessitates reasoning about the *collective* properties of groups of drugs and organisms. Since the utility of individual actions depends strongly on the other actions to be selected at the

same time, the strategy rule representation could not capture the desired drug-selection expertise. (If every possible collection of drugs was represented as an "superaction," then strategy rules could represent drug-selection criteria. However, this is not feasible for large numbers of drugs, and it reduces all strategic reasoning to a single decision.)

Actions can be related directly to the situations in which they should be chosen.

A positive example of a task with this property is selecting legal cases for argument, where cases are treated as actions. The merits of each case can be represented with features that describe its individual properties and its relationships to other cases and the current fact situation (e.g., Ashley's (1987) *dimensions*). A case-selection strategy might be modeled by relating the relevant features of legal situations to the features of cases that may be cited in the specified situations. For instance, in a trade secrets situation one might cite cases that make a claim about whether and how secrets were disclosed.

A negative example is the management of cancer treatment plans, the domain of the performance system ONCOCIN (Tu et al., 1989). The strategy for cancer treatment in ONCOCIN is represented with *protocols*: skeletal plans that are instantiated with therapeutic actions for particular patients. In attempting to model the individual treatment steps as actions in ASK, we found that the justifications for choosing the next action in cancer protocols were often statements of the form "because drug V is a member of the drug combination VAM, which is the next chemotherapy to be administered to this patient according to protocol 20-83-1" rather than "VAM is useful for small cell lung cancer because this combination can help prevent the tumor becoming resistant." The justifications also did not include a description of the context in which drugs V, A, and M are competing with other possible drugs. The knowledge underlying the recommendation of the VAM drug combination is compiled into the skeletal plan for a protocol. In this domain it is unrealistic to expect the experts to justify treatment plans with underlying reasons for their use, because by their very nature protocols are experiments designed to *test* the effectiveness of treatment strategies.

In general, the opportunistic style of control afforded by ASK's representation *generates* plans based on underlying reasons for taking plan steps (actions), when they are available. A memory-based planner unfolds and *instantiates* stored plans, in which individual actions need no independent justification. Applications requiring domain-specific strategic knowledge often do both styles of reasoning about action. *Within* an ONCOCIN protocol, for example, actions may be modified or dropped for reasons relating to the dynamic situation (e.g., the condition of the patient). Strategic knowledge for modifying steps within a plan could be formulated in strategy rules and acquired with ASK if the position of an action in a plan were abstracted as a control feature. In ONCOCIN this knowledge is, in fact, represented with rules that are indexed by protocols.[1]

---

[1]Thanks to Lawrence Fagan, Mark Musen, and Samson Tu for their help with this analysis.

Local action-selection criteria can avoid global pitfalls.

Computer players of adversarial games often are based on static evaluation of position. Their strategy for selecting a next move is to choose the action that scores best on the evaluation function. If the evaluation function can be structured as a conjunctive expression over features, ASK could be used to acquire it. For example, ASK can acquire the kind of strategy learned by Waterman's poker player: mappings from descriptions of the board and the opponent to betting actions (Waterman, 1970). A game-playing strategy based on mappings from features of game situations to classes of moves will succeed if the features are usefully predictive — if what looks good locally does not lead to global pitfalls.

A borderline negative example is chess, where strategy is often played out over several moves and evaluation functions are prone to horizon effects. If the right features can be found, strategy rules can map them to actions and ASK can acquire them. If the features invented by the user lead to pitfalls, then acquiring rules that use these features will not produce a globally optimal strategy.

In general, strategy rules support the reactive style of reasoning, where features are immediately available. In contrast, search-based planning can explore the outcomes of actions into the simulated future and back up the evaluation of the utility of the results. Therefore, ASK can be useful for tasks in which the effects of actions cannot be accurately predicted. The features acquired by ASK combine predictions of effects and the expected utility of effects.

An optimal decision among actions is not required or possible for every choice of actions.

The chest pain application is both a positive and negative example. Most of the evidence-gathering questions, tests, and therapies are chosen with relatively simple measures of utility, such as qualitative measures of diagnosticity, efficacy, and cost. In practice, the data and necessity to elicit probabilities and numeric estimates of utility for every possible combination of actions is not present. However, a negative example in the same domain is the *last* strategic decision that is typically made (or avoided): deciding whether to perform angiography and consequently open heart surgery. This decision has been successfully modeled using the techniques of decision analysis (Pauker & Kassirer, 1981).

There is no reason in principle why ASK's model of selecting actions cannot be described in terms of expected utility, nor is there any fundamental reason why a Bayesian utility function could not be used as a feature in strategy rules. The practical difference is in how a utility model is constructed. A set of strategy rules form a qualitative model of the utility of actions, where the union of actions recommended by rules are treated as equivalent. A multiattribute decision model (Keeney & Raiffa, 1976) makes finer-grained, numeric estimates of the relative utility of each attribute, and combines them to rank the recommended actions.

2.6.2 Critical assumptions

ASK makes progress in automating the acquisition of strategic knowledge, but many aspects of this difficult problem are not solved. What is left for further work

is revealed by the assumptions that the ASK approach makes about the available knowledge and the people that can provide it. Some key assumptions are discussed here, and a more complete list is in (Gruber, 1989).

Requirements on the substantive knowledge

The ASK approach assumes that substantive knowledge of the performance system: 1) is already acquired or can be acquired, 2) is correct, and 3) is sufficient for making the distinctions necessary for the strategic knowledge.

The control features used by ASK depend on existing substantive knowledge in the inference network of a MU performance system. For example, in diagnostic tasks, much of the important substantive knowledge is found in combination functions which specify how evidential support values and other inferential values are propagated through the inference network. In the MU environment, combination functions are acquired with a symbol-level interface — editors that present and elicit knowledge in the same form as it is used (i.e., rules, slot values, etc.). ASK assumes that the MU interface is adequate for acquiring substantive knowledge.

A more serious problem is the assumption that the substantive knowledge is correct. ASK's credit assignment algorithm determines what type of rule to acquire and which objects the rule must match and not match. The algorithm is based on the assumption that the features mentioned in existing strategy rules are correct. To account for the discrepancy between system and user actions, a new rule must match different features or different values of features than the existing strategy rules. If the features return incorrect values for some actions, this algorithm cannot correctly attribute the blame.

Finally, ASK assumes that the features that are already defined or are easily defined within the existing knowledge base are sufficient for representing the desired strategy. The experiment in reimplementing NEOMYCIN described in Section 2.5 was an opportunity to test this assumption. NEOMYCIN's strategy makes heavy use of the subsumption relation among hypotheses. For example, one metarule specifies that "If the hypothesis being focused upon has a child that has not been pursued, then pursue that child." The CHILD metarelation assumed by this rule is a subsumption relation among hypotheses that was not present in the MUM knowledge base used in our experiment. It was simply not possible to acquire this strategic knowledge without reorganizing the substantive knowledge base (i.e., identifying abstract categories of diseases and relating them in a hierarchy to the existing diseases).

In general, the overall effectiveness of ASK in acquiring strategic knowledge is bounded by the difficulty of representing the relevant control features for the domain.

Validity of expert's justifications for acquiring strategy

It was argued in Section 2.1 that the acquisition of strategic knowledge is difficult because domain experts do not normally express their strategy in a form that is generative, operational, and general (i.e., because of representation mismatch).

However, it is observed that experts *can* give justifications for specific strategic decisions. The approach taken in ASK requires a strong assumption: that experts' justifications form a valid basis for acquiring the strategic knowledge of systems. There are several ways that this assumption might be wrong.

One way is the problem of tacit knowledge — that the knowledge we wish to acquire from experts is not explicitly present in what they tell us. An influential theory in cognitive science argues that the knowledge underlying expertise is often tacit due to the process of *knowledge compilation* (Anderson, 1986). As experts learn problem-solving strategies from experience in a domain, they internalize the useful associations between situations and actions and become unaware of the inferential steps that they may have made as novices. For example, physicians in a educational setting may teach diagnostic strategy one way and practice it another way. In experimental settings, when people are asked to account for their decisions retrospectively they often refer to causal theories or judgements of plausibility rather than the pertinent stimuli and their responses (Nisbett & Wilson, 1977). And some writers argue that the difference between being able to act and being able to talk about action is fundamental — that computer models of action are essentially incapable of capturing the real basis for action (Winograd & Flores, 1986).

If experts cannot account for their strategic decisions, ASK cannot acquire the strategic expertise in a program. There is a difference, however, between assuming that experts can describe their own cognitive processes and assuming that they can justify their behavior. ASK only depends on the latter assumption. The assumption that experts can provide valid justifications may be reformulated as the requirement that experts be good teachers. Remember that ASK is designed to acquire knowledge for choosing actions that are observable, and therefore objectively justifiable. The fact that medical school professors may not practice what they preach does not mean that the justifications are invalid. On the contrary, good teachers can account for behavior in a principled way and in objective terms, even though their compiled expertise may not follow from their explanations.

A second problem with the reliance on expert-supplied justifications is the assumption that domain experts can invent useful abstractions of the domain — the right control features. In the same way that an autonomous machine learning program is limited by the description language provided by the program author, a knowledge acquisition system such as ASK is dependent on the abstraction skills of the user.[1] ASK relies on the user to invent features that not only are sufficient to distinguish actions in specific cases, but also lay out a space of relevant generalizations. This assumption would be unfounded if the expert defined a unique feature for every training case; the resulting strategy — a lookup table of special cases — would be brittle. It is also possible that an expert can describe useful features in natural language but cannot implement them.

---

[1]Getting the right primitive features has always been essential to getting a machine learning program to find useful generalizations. For example, Quinlan (1983) reports having spent three months devising a good set of attributes (board position features for chess) so that the learning program ID3 could produce a decision tree in seconds.

The validity of an assumption about the skill level of users is an empirical question, and the answers will depend on the subjects and the tasks. ASK helps frame the research question by distinguishing between the ability to *invent* the necessary features, which is structured by the elicitation of justifications, and the *implementation* of features, which is partially supported by a symbol-level interface for defining features. If an ASK user cannot implement a feature but knows what it should represent, she calls the knowledge engineer. The acquisition of features (new terms) is an interesting area for further study.

2.6.3 Major limitations

Two of the fundamental limitations to the approach taken in ASK are discussed in this section. A more complete analysis is given in (Gruber, 1989).

Reliance on knowledge engineering skills

It should be clear from the preceding discussion that ASK depends on the ability of the user to define and implement control features. The fact that many features are not easy to implement means that ASK is still limited by the operationalization aspect of representation mismatch. The problem of operationalizing terms is relevant to any learning system whose description language can be extended by the user. Although ASK provides a helpful interface for defining new features, some new features require programming to implement. The problem is not a matter of learning the notation; one needs to know a lot more than the syntax of Lisp to be able to implement control features that capture sophisticated assessments of the state of problem solving. To implement a feature such as the potentially-conclusive-evidence relation, one needs to understand the workings of the MU architecture at the symbol level. That is the expertise of knowledge engineers, not domain experts.

There is a way in which ASK's elicitation technique can actually aggravate the problem of representation mismatch. ASK is designed to present the "user illusion" (Kay, 1984) of an interface that accepts *explanations* for strategic decisions. In contrast, a symbol-level acquisition tool such as TEIRESIAS (Davis, 1976) supports a straightforward interface to rules without disguising them as anything else. The problem with a system such as ASK that presents a knowledge-level interface to the user but internally makes symbol-level distinctions is that the user's model of how the system works can differ significantly from how the system actually functions. If the user's model is inaccurate, she cannot predict what the system will do with what is elicited. The result is a breakdown in communication and a failure in the knowledge acquisition process.

One of the experiments in which ASK was used by physicians illustrates a case in which the user's ignorance of the operational semantics of strategy rules resulted in an unintended strategy. The expert wanted to teach the system to ask all applicable questions of one class before asking any applicable questions of another. He answered ASK's prompts in such a way that the credit assignment algorithm determined that it needed to acquire a filter rule, when in fact a selection rule was needed. When the expert explained (with justifications) that questions of one type should *not* be selected, ASK generated a filter rule that

prohibited questions of that type from *ever* being selected, which is a gross overgeneralization. The error was not apparent until the actions from the first class were exhausted and the system could not suggest any more actions to perform. To have avoided this problem, the user would have had to understand the operational difference between filter and selection rules and the correspondence between his answers to ASK's prompts and the type of rule being acquired.

Overgeneralization due to the lack of a training set

Although ASK uses generalization operators, it differs from most inductive learning techniques in that it does not learn from a large training set of examples. The user is responsible for choosing training examples that will produce useful generalizations. Unfortunately the lack of a large training set limits the extent to which ASK can help with the generalization problem.

It is easy to generate strategy rules with ASK that are overly general, because of the elicitation technique. Adding justifications specializes the resulting strategy rule; doing nothing leaves it general. Consider two strategic situations in the medical workup. In the early phase, actions are selected for their low cost and minimal diagnosticity. In later phases, actions that offer a potentially significant diagnostic or therapeutic value are selected at higher cost, even if lower cost actions are available. If the selection rules for the first phase were acquired without any clauses identifying the strategic situation (i.e., features of the early phase), then the rules acquired for the early phase would also match when the later phase arose. There is no knowledge-free way for ASK to anticipate the missing clauses that specify the context in which a rule should apply.

In practice, overgeneralizations of this type are discouraged by starting with an initial set of strategy rules that specify the basic strategic situations to distinguish. These rules serve as the basis for seed justifications (Section 2.4.5) upon which the user builds a set of justifications for a specific case. The knowledge engineer can provide a set of very general strategy rules, anticipating some of the situations in which domain-specific tradeoffs will arise. Then the major role of the user is to *specialize* the general strategy with application-specific strategic knowledge. Overgeneralizations are still likely, however, when user fails to elaborate the features of a novel context in which a selection is made among specific actions.

If ASK kept a library of training cases, it might be able to check newly-formed rules for inconsistency with past training and prevent excessive overgeneralization. Each case in a library would need the values of all relevant features of the positive and negative examples and the features specifying the strategic situation. When a new rule is proposed, it could be tested against the objects in the case. If the new rule recommended a different outcome than the stored case, and did not shadow the rule associated with the case, then the two rules would be inconsistent. Unfortunately, keeping a library of cases is not trivial because the space of features can grow with experience. If a new rule mentions a new feature, it is incomparable with previous cases that did not mention the feature, unless the feature is static (i.e., its value does not change during the execution of the performance system). A general solution is to store a snapshot of the entire working memory with each case, so that all possible relevant features

could be derived. This solution could be expensive. The whole issue of how to store experience for future learning is an intriguing avenue for research. Some promising approaches have been developed for case-based learning systems (e.g., Bareiss, 1989; Hammond, 1989).

2.7 Discussion: Key design decisions

Design decisions are often hidden sources of power in AI systems. This section discusses a few characteristics of ASK's design as they relate to its function as an automated knowledge acquisition tool.

The strategy rule representation supported by ASK is neither a novel way of formulating strategy nor an ad hoc design. For the purpose of *implementing* strategic knowledge, a procedural representation such as a Lisp function or an augmented transition network would have been more flexible. The goals in designing a representation for ASK are to be able to capture strategic knowledge in an executable form and to be able to elicit it from experts.

Strategy rules were designed to represent mappings between states of the inference network and equivalence classes of actions, for each of three operations: propose, filter, and select. The declarative clausal form of strategy rules allows for execution by conventional unification-style matching and also corresponds to the structure of justifications. Limiting the operational effects of rules to propose, filter, and select operations simplifies credit assignment and conflict resolution. The result is a representation in which strategic knowledge can be acquired.

Two of the design decisions that led to this representation are critical to ASK's techniques for automated knowledge acquisition. First, strategic knowledge has been formulated as classification knowledge. Second, a global strategy is represented as a family of strategy rules with fine-grained effects. The rationale for each decision is given below.

2.7.1 Formulating strategic knowledge as classification knowledge

Strategy rules structure knowledge about what to do next as knowledge for classification: associations between strategic situations and classes of actions. The following capabilities follow from this design.

The ability to use conventional machine learning techniques.

ASK can use simple syntactic induction operators for generalization (turning constants to variables, dropping conditions, and extending reference). Whereas the problem of learning sequences and procedures with internal state is very hard (Dietterich & Michalski, 1986), the problem of learning classification rules is well understood (Dietterich and Michalski, 1983). If mappings from states to actions define the classes of state descriptions in which actions are appropriate, a learner can generalize control knowledge by generalizing class descriptions.

The ability to elicit machine-understandable information at the knowledge level.

ASK can elicit applicability conditions for control decisions in machine-understandable terms, because the justifications from the user's point of view correspond to clauses in the rule representation. The list of justifications can be elicited in any order, since they are used as conjuncts in the class description.

The ability to use simple explanations for input and output.

ASK can use simple template-based natural language generation to provide explanations. ASK's explanations are just lists of facts relevant to the current control decision paraphrased in English; they are essentially the same as justifications. ASK can get away with this simple explanation technique because every control decision is a flat match of situations and associated actions. Because there is no implicit state, such as in an evolving control plan, the context of the decision to choose an action is fully explained by the clauses of matching strategy rules. The English explanation — paraphrases of instantiated clauses — corresponds to what is happening at the symbol level.[1]

The use of explicit, abstract control knowledge for explanation was developed in the work of Swartout et al. (Swartout, 1983; Neches, Swartout, & Moore, 1985) and Clancey (Clancey, 1983a, 1983b). ASK follows the principle arising from their work that an explanation of surface behavior should correspond to the structure of the system's strategy. However, in contrast to serious attempts at knowledge system explanation, ASK's explanations do not describe the goal structure and focusing behavior of the system because the performance architecture does not support the corresponding control mechanisms (e.g., goal stacks, tasks, etc.).

The inability to acquire goal-directed plans.

As a consequence of formulating strategy as simple classification, it is awkward to acquire goal-directed strategy with ASK. To capture the knowledge for reasoning about action at different abstraction levels, the strategy rule representation would have to be extended to support hierarchical planning in the sense of ABSTRIPS (Sacerdoti, 1974). Currently, all strategy rules within each category (propose, filter, select) are matched in parallel at each iteration. In one extension proposed in (Gruber, 1989), the rules would be partitioned into abstraction levels; at each level, rules would choose the subgoals for the lower abstraction level until the subgoals at the lowest level are grounded in individual actions. It is not clear whether the added structure would compromise the comprehensibility of the elicitation technique; this is a question for future research.

2.7.2 Formulating strategy as fine-grained reactions

Recall the third aspect of representation mismatch: domain experts have more difficulty devising a general procedure that accounts for their strategic expertise

[1]This an oversimplification. In actuality, the shadowing relations among strategy rules are not reflected in the explanation. Not surprisingly, they are a source of confusion for users, possibly because they do not fit the simple conceptual model of situation-action.

than describing what they actually do in specific cases. ASK shows that strategic knowledge can be acquired from experts if it is elicited in the context of specific choices among actions and then generalized. This is possible because strategy rules model local decisions about actions that can be generalized to classes of situations and actions. In theory, what appears to be a global strategy can emerge from a series of local strategic decisions. For example, Chapman and Agre (1986) propose that complex, coherent behavior arises from the continued activation of situation-action structures without top-down control.

There is empirical support for the notion that globally coherent plans can be acquired by eliciting the knowledge for local decisions. For example, SALT succeeds at acquiring knowledge about how to construct globally satisfactory solutions to a class of design problems (Marcus, 1987, 1988). SALT elicits from designers knowledge about constraints among individual parts — information that is relatively easy to specify — and offers help for putting the pieces together. SALT's results are relevant to ASK because constructing a solution requires managing the *process* by which parts are assembled under constraints; this is similar to managing the selection of actions. SALT can acquire the requisite knowledge from experts because it decomposes the larger task of assembling a solution into small decisions about what part to add, how to (immediately) check it for constraints, and how to recover from those violated constraints.

One can view SALT's design task and ASK's action-selection task as varieties of planning, where configured parts and diagnostic actions correspond to plan steps. This view reveals an important difference between the two architectures. SALT's planning method provides for a backtracking search, whereas ASK's planning method is purely reactive, with no projection (lookahead) and no possibility to undo actions. This may prove to be an important variable in the question of whether knowledge of local decisions can add up to a global strategy.

2.8 Conclusion

The immediate outcome of this research is a method for partially automating the acquisition of strategic knowledge from experts. The issues that are raised, however, are more significant than the ASK program itself. Strategic knowledge was chosen for the study of knowledge acquisition because it illuminates the problems of representation mismatch. Furthermore, an extreme solution was selected — a declarative representation of reactive control knowledge — to test conjectures about sources of power for knowledge acquisition. The results have been analyzed in the preceding discussions of the scope of applicability, assumptions, limitations, and design decisions. This section concludes with a more general point brought out by this work and the future research it suggests.

If representation mismatch describes the problem of knowledge acquisition, then solutions should offer some way to bridge the representational gap between the domain expert and the implementation. This suggests that the design of knowledge representations is central to addressing the knowledge acquisition problem. This article has emphasized the motivations for and implications of ASK's representation of strategic knowledge in an effort to elucidate principles of

design for acquisition: how to design knowledge systems to facilitate the acquisition of the knowledge they need.

Earlier reports (Bylander & Chandrasekaran, 1987; Gruber & Cohen, 1987) describe how knowledge representations and methods for task-level architectures can facilitate *manual* knowledge acquisition (i.e., mediated by tools that are passive). The design of representations can reduce representation mismatch from the implementation side by providing (generic) task-level primitives which enable experts to work directly with the knowledge base.

The ASK research illustrates how *automated* knowledge acquisition can help overcome representation mismatch by eliciting knowledge in a form that is available from experts and yet is very close to an operational, generalizable representation. Again, the design of representations plays a central role in the success of the knowledge acquisition process. The major contributions of ASK to the process — active elicitation of justifications, credit assignment, and syntactic generalization — are enabled by the declarative, role-restricted rule representation. At the same time, the kind of strategic knowledge that can be acquired — opportunistic and reactive rather than goal-directed and plan-driven — is a function of what can be naturally represented in strategy rules.

A similar power/generality tradeoff can be found in most knowledge acquisition tools. At the power end of the continuum lie OPAL-class elicitation tools (Freiling & Alexander, 1984; Gale, 1987; Musen et al., 1987), which acquire knowledge for representation customized to a problem-solving method and a particular domain. OPAL employs elicitation techniques that are customized for both the skeletal-plan refinement method used in ONCOCIN and the domain of cancer therapy. As a result, OPAL can be used by domain experts. At the generality end lie TEIRESIAS-class tools (Davis, 1976; Boose & Bradshaw, 1987; Shachter & Heckerman, 1987), which acquire knowledge at the symbol-level for formalisms that are not committed to particular tasks or domains. TEIRESIAS makes it easy to enter and modify rules but requires the user to bridge the representational gap from the domain- and problem-specific description to the backward-chaining architecture. Somewhere in the middle are the MOLE-class tools (Eshelman, 1988; Klinker, 1988; Marcus, 1988), which acquire knowledge in representations that are method-specific and domain-independent. This article has shown several ways in which the design of ASK trades the generality of a representation useful for knowledge engineering for the power of a restricted representation suitable for automated knowledge acquisition.

Further research is needed to investigate how knowledge representations and reasoning methods can be designed to make the task of knowledge acquisition more amenable to computer-assisted techniques for elicitation and learning.

Chapter 3

A Declarative Representation of Control Knowledge

3.1 Introduction

Control is an important problem in AI: Knowledge systems are getting very large and difficult to control [DAR, 1987], [Erman and Lesser, 1975], and, because efficiency is a concern in these systems, control strategies must be flexible enough to balance various costs, especially time [Lesser *et al.*, 1988], [DAR, 1987], [Dean, 1987a], [Dean, 1987b], [Hanks, 1987], [Herman and Albus, 1987], [Korf, 1987], [McDermott, 1982]. Independent of efficiency concerns, we are beginning to work on tasks such as design [Howe *et al.*, 1986], [Orelup, 1987], [Orelup *et al.*, 1988], process control [Pardee and Hayes-Roth, 1987] and knowledge-based planning [DAR, 1987], in which "how to" knowledge is important. In these tasks, problem solvers do not use a single, fixed strategy, but change strategies as the situation demands, keeping "trim" to the current situation.

AI researchers are beginning to recognize control knowledge as a kind of expertise, and are developing tools to help knowledge engineers acquire it [Marcus *et al.*, 1985], [Marcus, 1987], [Bareiss *et al.*, 1987], [Boose and Bradshaw, 1987], [Eshelman, 1987], [Musen *et al.*, 1987].

Previous work by Tom Gruber in our laboratory has addressed the problem of acquiring and generalizing strategic "meta-rules" [Gruber, 1988a], [Gruber, 1988b] similar to those discussed by Davis [Davis, 1976], and

Clancey [Clancey,], [Clancey, forthcoming]. Meta-rules give control a "one step at a time" or reactive flavor that, when implemented in a medical expert system [Cohen *et al.*, 1987a], [Cohen *et al.*, 1987b], failed to capture some aspects of diagnostic expertise [Cohen and Day, 1988]. For example, it was difficult to formulate meta-rules that had contingent actions on their right-hand sides—to say "do test A and if the answer is positive do test B otherwise do test C." The current work was initiated to develop representations for these little contingency plans, but it swiftly became an exploration of representations for the range of knowledge one needs to control complex AI systems. Throughout, we have required these representations to be declarative, so they may be accessible to knowledge engineers, and also to knowledge acquisition tools.

This paper presents a snapshot of our current work on control. It represents work in progress and so poses problems that it doesn't solve. As AI researchers explore more realistic environments, we think it is essential that they report where they are, how they got there, and where they are going; even if they still have a long way to go [Cohen and Howe, 1988b], [Cohen and Howe, 1988c]. In this spirit, we have organized the paper into six sections that correspond roughly to aspects of a journey:

We begin with an analysis of the control literature that led us to the idea of strategy frames—declarative structures that represent problem-solving strategies (Sec. 3.2). Next, we discuss the purpose of the journey, our intent to empirically test some hypotheses about local and global control via strategy frames (Sec. 3.3). In the next two sections we describe a process control task and an implementation in terms of strategy frames which, together, provide the environment for our empirical work (Secs. 3.4 and 3.5). Preliminary results are described in Section 3.6.

3.2 Strategy frames: a view of control

Our view of control is motivated by the following observations, which are based on analyses of the control strategies of CASNET [Weiss *et al.*, 1977], [Weiss *et al.*, 1978], PIP [Pauker *et al.*, 1976], HEARSAY-II [Erman and Lesser, 1975], [Erman *et al.*, 1980], MUM [Cohen *et al.*, 1987a], MDX [Chandrasekaran *et al.*, 1982], [Chandrasekaran and Mittal, 1983], NEOMYCIN [Clancey, 1986], [Clancey,], DOMINIC and DOMINIC-II [Howe *et al.*, 1986], [Orelup *et al.*, 1988]:

Observation-1: The control strategies of many knowledge systems can be viewed as the interaction of a small number of simpler strategies. We call these *strategy frames.*

The structure of strategy frames will be discussed in Section 3.5. For now, one can imagine them controlling inference within and between levels of *hypothesis spaces.* For example, CASNET's hypothesis space had three levels, for data, pathophysiological states, and disease hypotheses, respectively (Fig. 3.1). Its control strategy can be viewed in terms of the interaction of three strategy frames: bottom-up inference from data to pathophysiological states, lateral inferences along causal pathways between pathophysiological states, and bottom-up inferences between pathophysiological states and disease hypotheses. The tails and heads of the arrows in Figure 3.1 are different regions of the hypothesis space and are the domain and range, respectively, of strategy frames.

Observation 2: Strategy frames do not relinquish control after every inference, nor do they keep control throughout problem-solving.

A strategy frame says, for instance, "I'm in control, and we're going to do some bottom-up processing until it appears that some other kind of processing would be more useful." For example, MUM was controlled by *strategic phases* that changed relatively infrequently. A control cycle in MUM begins by looking at the state of the hypothesis space to determine whether the applicability conditions of the current strategic phase still obtain. If not, a new strategic phase is invoked. In either case, the next step is to select a focus of attention, and then to select evidence for or against that focus. The evidence is solicited and the state of the hypothesis space is updated. Then the cycle begins again. The system can stay in a strategic phase (i.e., under the control of a single strategy frame) for many cycles. For example, one strategic phase—called **Deal-with-Critical-Possibilities**—was active as long as critical, dangerous hypotheses (e.g., heart attack) had some degree of support. Within this phase, MUM selected hypotheses to be its focus of attention in order of their criticality; and it first sought low-cost evidence against the focus of attention, but had no prohibition against high-cost evidence pro or con the focus. Unlike other strategic phases, the **Deal-with-Critical-Possibilities** strategy gave MUM's problem solving a distinct "this is important, so hang the cost" flavor for the duration of time that critical hypotheses were active.

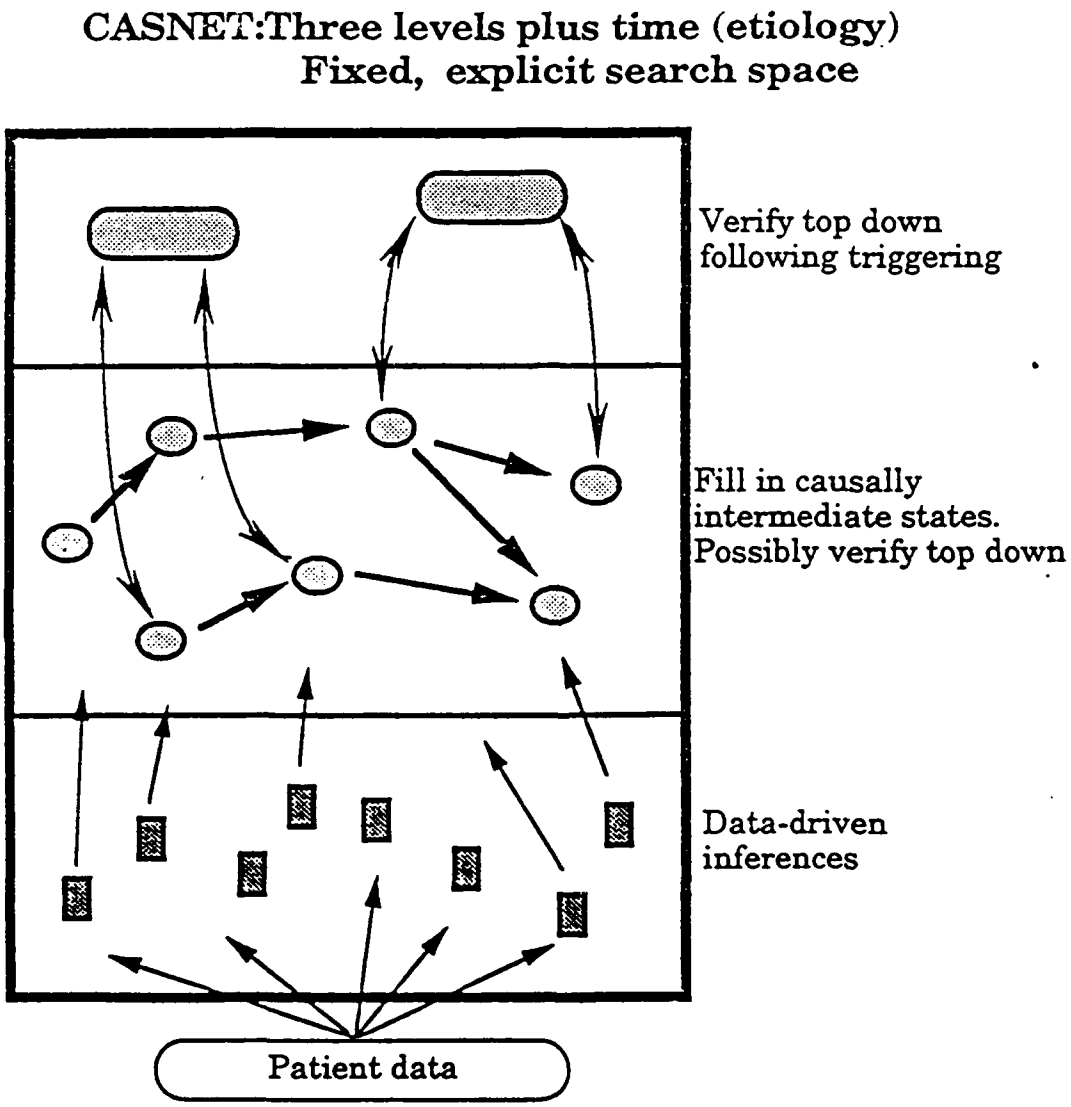


Figure 3.1: A schematic representation of control in CASNET, showing the levels of the hypothesis space and the strategy frames represented as vectors.

One is tempted to equate knowledge sources in HEARSAY-II with strategy frames. But although the stimulus and response frames of knowledge sources (KSs) in HEARSAY-II are analogous to the domain and range of strategy frames, the latter take control of processing for intervals that can involve many inferences, whereas KSs generate knowledge-source instantiations (KSIs) for each possible inference and relinquish control to the scheduler after every inference.

In fact, the designers of many systems have found it desirable to give them something like the functionality of strategy frames, even when they were initially designed to have opportunistic control, or at the other extreme, completely fixed control. HEARSAY-II was designed to have opportunistic control, but was later modified to have two phases—a bottom-up phase followed by an opportunistic one. Even MYCIN, which is commonly thought to be an exhaustive backward chaining production system, switched to limited forward chaining when it was presented with particular kinds of data [Shortliffe, 1976].

> **Observation 3:** Strategy frames are nested structures, in which there are *tactical instantiations* of the components of a strategy.

All strategic phases in MUM have the same nested structure:

1. Applicability conditions
2. Criteria for selecting focus of attention
3. Criteria for selecting evidence

but they differ in how the components or slots of the strategies are instantiated. For example, the criterion for selecting focus of attention in the **Discriminate-Strongest-Hypotheses** strategic phase is plausibility; this phase focuses on hypotheses that are likely given the evidence. In contrast, the **Deal-with-Critical-Possibilities** phase focuses on hypotheses that are dangerous and have some level of support; these hypotheses may actually have a low plausibility.

A similar view is found in DOMINIC-II, a program for *iterative redesign* of mechanical devices. The program has a five-step basic control cycle:

1. select an aspect of the design to improve
2. determine how much improvement is desired

3. select a design variable that, when changed, is expected to improve the design

4. determine how much to change the design variable

5. decide whether or not to change the design variable

For example, DOMINIC-II may want to improve the *expected life* of pulley system (step 1), from "short life" to "medium life" (step 2), by changing the diameter of the drive pulley (step 3), from four to five inches (step 4). If this change is predicted to have the desired effect, and it improves the overall evaluation of the design, then it will be adopted. Then the redesign cycle starts again. Roughly, redesign in DOMINIC-II is hill-climbing because each change to a design improves its overall evaluation. But it isn't strict hill-climbing because, depending on the tactical instantiations of the five steps in the redesign cycle, DOMINIC-II can actually allow a design to get worse before it gets better. For example, one tactical instantiation of DOMINIC-II's basic design strategy is:

1. **select an aspect of the design to improve:** select the aspect that has the *largest negative effect* on the overall evaluation of the design.

2. **determine how much improvement is desired:** require an improvement sufficient to ensure that the aspect no longer has a negative effect on the overall evaluation of the design.

3. **select a design variable that, when changed, is expected to improve the design:** select any design variable that has not been changed in the last two cycles.

4. **determine how much to change the design variable:** change the value of the design variable "a lot".

5. **decide whether or not to change the design variable:** even if the overall quality of the design decreases, accept the change if it improves the specific aspect of the design as desired.

DOMINIC-II monitors the current state of its design, looking for pathological situations. When it finds one, it typically switches from one tactical instantiation of the basic redesign strategy to another, more appropriate one. For example, a common problem in hill-climbing is the *mesa effect*: instead of moving steadily up a hill, a system gets trapped in a relatively flat area,

making many small changes but not improving overall performance. When DOMINIC-II detects this situation, it adopts a tactical instantiation of the redesign strategy that makes very large changes to a design variable even if they reduce the quality of the design. This is like taking large steps instead of little ones to get off the plateau onto a hill—even if one lands on the hill below one's current altitude. In a series of experiments we found that Dominic-II, which could select among tactical instantiations, always outperformed an earlier system, Dominic-I, which could not [Orelup *et al.*, 1988].

Observation 4: The same strategy frames show up in many knowledge systems, though parameterized differently.

In the last few years there has been a sense that many AI tasks are very similar. This sense was given voice by Clancey's characterization of diagnostic reasoning [Clancey, 1984b], [Clancey, 1985], and by Chandrasekaran's evolving taxonomy of AI tasks [Bylander and Chandrasekaran, 1987], [Chandrasekaran, 1986], [Chandrasekaran, 1983]. It has been argued that *task-level* architectures can be designed that are more general than particular knowledge systems but less general than weak methods such as generate and test [Gruber and Cohen, 1987b], [Gruber and Cohen, 1987a]. What makes AI tasks similar is not the facts and heuristics we use to solve them, for these vary from one domain to another, but rather the general *kinds* of knowledge they require and, most important from our perspective, *how* they are solved. In our analysis of many knowledge systems, we believed we repeatedly saw the same strategies. For example, all the diagnostic systems made some distinction in their control strategies between data that "trigger" hypotheses and those that cannot trigger hypotheses but can support previously triggered ones. Most of these systems also made their control strategies sensitive to data that could categorically rule out hypotheses. The basic control strategy for diagnosis, though slightly different in each of the several systems, was to first use a subset of the data to generate a small set of hypotheses (using *all* the data would create a combinatorially unmanageable set), then to try to rule out or rule in these hypotheses, typically in an order that reflects the importance of the hypotheses.

The advantage of identifying general strategies is that they become part of the knowledge engineer's toolbox. We imagine providing task-level architectures complete with a variety of declarative strategy frames, each easily parameterized for the particular application, much as knowledge-engineering tools currently provide declarative representations to be filled with domain-specific facts and heuristics. Several steps have already be taken in

this direction [Gruber, 1988a], [Marcus *et al.*, 1985], [Bareiss *et al.*, 1987], [Eshelman, 1987], [Musen *et al.*, 1987].

Observation 5: Control strategies can depend intimately on the structure of the hypothesis space.

CASNET's control strategy exploited causal associations among pathophysiological states, and MDX's strategy exploited hierarchical associations in a taxonomy of diseases. It seems that the diversity of control strategies depends on how many types of relations exist among objects in hypothesis space. For example, if the only possible relation in the hypothesis space is **evidence-for**, then a system is limited to blind data-directed or goal-directed control. It can't focus on hypotheses that are causally related to other likely hypotheses unless causal relations are explicit in the hypothesis space. In NEOMYCIN, Clancey describes many relations that are needed to support a wide range of diagnostic subtasks. These include binary relations (e.g., causal and hierarchical) relations, and also unary relations or properties of the objects in hypothesis space. A similar approach was taken in MUM and the subsequent MU project. In MU, one defines sets of objects based on their relations with other objects, such as *the set of all tests that potentially-confirm any object in the differential and are inexpensive.* Here, *potentially-confirm* is a binary relation and *object in the differential, test* and *inexpensive* are unary ones. Sets in MU can function either as foci of attention or, as in this example, as sets of potential evidence.

In PIP, the relationship between control—specifically focus of attention—and the structure of the hypothesis space is extremely tight. The hypothesis space is a network of associated frames, most of which represent diseases. These frames are "activated" or "illuminated" when their associated symptoms are found in the patient. Hypotheses become active (i.e., part of the focus of attention) when activation spreads over relations in the hypothesis space. For example, PIP has triggering relations between data and hypotheses that make hypotheses active if the data are present. A more complex role is played by relations such as *may be caused by.* If two frames are associated by this relation, and one becomes active, then the other becomes *semiactive*, which means, roughly, that it will have a greater propensity to become active as more data become available.

It is easy to see the importance of the structure of the hypothesis space when that structure is *explicit*, as it is in CASNET, MDX, NEOMYCIN, and MUM. By explicit we mean that all data, intermediate hypotheses and conclusions are known in advance before the system is ever run. In contrast,

objects in *implicit* hypothesis spaces are generated by search during execution. For example, in the DOMINIC systems, we do not traverse an explicit space of thousands of designs, but rather we generate the space by iteratively modifying each design to produce the next. In such cases, objects in the hypothesis space are not associated by explicit relations, as they are in explicit hypothesis spaces. Consequently, control strategies are designed for the *implicit structure* of implicit hypothesis spaces. In DOMINIC, this structure was assumed to be a hill, and so design was viewed as hill climbing. In fact, DOMINIC-II is able to detect the local topology of the hill and, if it is a plateau, modify its control strategy appropriately. In HEARSAY-II, the implicit structure of the space of interpretations was assumed to contain many constraints—often referred to as redundancy in the speech signal—so that partial interpretations of one part of the signal could help the system interpret other parts. Like DOMINIC-II, HEARSAY-II could detect where it was in the space and could extend relatively certain regions into less-certain areas. This was called island-driving.

3.3 Motivations

Although AI researchers have been building control structures and problem-solving strategies for years, one of the basic questions about control remains unanswered: Under what conditions can you achieve a sequence of actions that look like they were selected by a *global* strategy, when in fact they were selected by one or more *local* strategies? Both global and local are vague and at best relative terms: one strategy relies on "more global" information than another. Informally, local means *based on a subset of the available information.* In terms of performance, because local strategies require less information and less integration of information, they are valuable when the cost of obtaining and processing relatively global information is prohibitive. On the other hand, performance is typically better when informed by global information. For example, imagine picking out a route across a city: a relatively local strategy determines the route given the global goal and the immediate environment, whereas a more global strategy considers the environment further away. The local strategy requires less information and less planning, but may take us away from our goal and into "blind alleys"; the more global strategy can avoid these problems because it has access to more global information, such as a map. In terms of this example, we want to know under what conditions one can traverse a route that *appears* to be

guided by a map when it isn't.

Our research poses this question not in terms of actions such as traversing a route, but in terms of strategies (implemented by strategy frames) that select actions: Under what conditions can a sequence of strategies appear to be guided by global information when, in fact, it isn't? The question is important because AI is working in task domains that seem to require multiple strategies (or, at least, multiple tactical instantiations of strategies). Examples include Dominic-II and MUM (Sec. 3.2), as well as recent work suggesting that alternative strategies should be selected by resource demands and availability [Lesser *et al.*, 1988], [Garvey *et al.*, 1987], [Pardee and Hayes-Roth, 1987]. We want to know the conditions in which a system with multiple strategies needs global information to select them, and, conversely, when relatively local information will suffice. Under what conditions does the computational cost of global information outweigh the benefit of having it? Under what conditions does relatively local selection of strategies result in inefficiency, incoherence, or other pitfalls analogous to "blind alleys" in the example above?

Our research is designed to explore these questions. Initially, we thought about reimplementing some of the systems discussed earlier—CASNET, PIP, MUM, and so on—with strategy frames. But on reflection it seemed uninteresting to demonstrate yet another architecture for diagnosis. Instead, we have used strategy frames to control a system that solves a relatively new and uncommon kind of task: *Process control* tasks require a system to monitor and adjust to variations in an ongoing process, many or all of which are unpredictable.

3.4 The McD Problem

McD is a simplified model of a fast-food restaurant. Orders are presented at varying rates over time. McD tries to fill all orders as quickly as possible without building up large surpluses of items. It does this by changing the rates at which it produces items—by shifting employees from one activity to another. For example, if McD has a surplus of hamburgers but a shortage of shakes, an employee may be moved from the grill to the beverage station.

The McD problem has these characteristics:

Dynamic demands—the problem solver must respond dynamically to changing demands; for example, changes in the rate at which orders are placed.

Resource allocation—problems are solved by dynamically shifting resources from one activity to another, thereby changing the configuration of the problem solver; for example, by moving employees to different stations.

Tradeoffs—even if a system has the resources to handle any level of demand, it shouldn't want them to be committed when demand is slack, since they would be largely unproductive. There is a tradeoff between the costs associated with the failure to meet demand and those associated with excess supply. And this is but one of many tradeoffs that together determine the *efficiency* of a system.

In McD, resources are employees and products are menu items (hamburgers, fries, soda, shakes, apple pies) and service items (cashiers, clean tables, and condiments). The number of employees is variable, but in our experiments is initially seven. Employees work at stations. There are two grills (each with space for two employees), two fryer stations (one employee tends each), one drinks and dessert station (tended by one employee), two busing stations that clean tables and manage condiments and utensils (tended by one employee each), and four cashier stations (each tended by one employee). Obviously, more than seven employees are needed to fully staff all the stations in McD. Successful management of McD involves shifting employees from one station to another in response to demands. McD can also call in additional employees and send surplus employees home.

McD is presented with blocks of orders at varying rates. A block includes the number of people in a group, whether they require tables, and how many of each menu item are desired.

Since cashiers take orders in McD, and the number of cashiers is limited, some orders will not be processed immediately. Moreover, not all menu items are available all the time, so some orders will not be filled immediately. Two components of McD's performance are the average time waited for a cashier and average time waited for food.

McD recognizes three special *situations*: Shortages, surpluses, and breakdowns. A shortage is indicated when the expected demand for an item significantly exceeds the number on hand (minor disparities between demand and supply are ignored). Conversely, a surplus is indicated when the expected demand is significantly less than the number on hand. (Expected demand is just the average demand over the last few time segments.) Breakdowns remove stations from commission until resources are allocated to fix them. For example, one of the two grills may go down and may result in a shortage if demand is high and it isn't fixed quickly.

McD has strategies for each of these situations. All involve moving people from one station to another. A shortage of hamburgers, for example, is handled by searching for an employee who can stop what he is doing and move to a grill. Of course, if the capacity of the grills is exceeded (i.e., more than four people are already working them) then McD can do nothing to reduce the shortage. Thus, shortages of beverages and desserts cannot be overcome unless these stations are unmanned, because each of these items is produced at a station that has a capacity of one employee. Breakdowns are fixed by moving an employee to the broken station to fix it. Surpluses, conversely, are rectified by removing an employee from a station.

We must digress briefly to explain what is strategic about problem solving in McD. Colloquially, strategies select actions and remain in effect longer than individual actions. By strategy, we mean a configuration of the problem solving system (McD) that remains in effect over many problem solving actions. For example, McD may fill many customer orders with just two employees working at a grill, but eventually a shortage may require a change in this configuration. Adding another person to a grill will change McD's behavior: it may now produce a surplus of hamburgers, or experience shortages of other items, or the waiting period for food may decrease, or it may increase due to a bottleneck introduced by removing the employee from his previous station. The strategy frames described below change McD's behavior by changing its configurations. A similar notion of strategy is found in, say, DOMINIC-II (see above), where different strategies produced different hill-climbing behaviors.

In sum, the McD problem is to dynamically monitor and alter configurations, that is, allocations of resources to stations, so that shortages and surpluses are avoided. McD detects shortages, surpluses, and breakdowns, and selects among strategies for rectifying these situations.

3.5 Strategy Frames

Strategies are represented by strategy frames. Figure 3.2 is a strategy frame called **shuffle-resource-to-menu** (**shuffle** for short). The figure shows an instance of **shuffle** that was invoked somewhere in the middle of a run of McD.

shuffle is invoked to fix shortages in menu items, which include shortages of hamburgers, fries, soda, shakes, and apple pies. **shuffle** has an **applicability** slot that contains a lisp function called `is-there-a-food-shortage`,

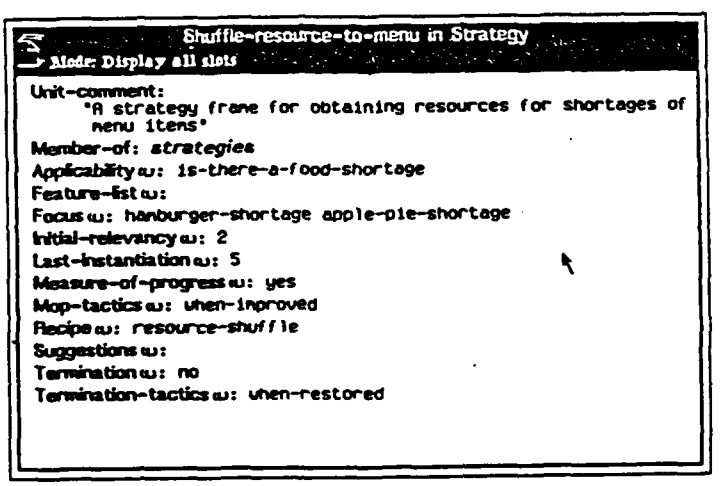


Figure 3.2: Strategy frame for moving resources to cover shortages of menu items.

which the interpreter runs to determine whether there is a shortage of any food items. If the applicability condition is met, the interpreter adds **shuffle** to the list of applicable strategies. Typically, this list contains several strategies, ranked by their **relevance** slots. Relevance can be calculated many ways, that is, the relevance slot can have many tactical instantiations. But in **shuffle** and other strategy frames that deal with shortages, only one tactical instantiation is currently used (others are planned but not yet implemented). It ranks strategy frames by the number of known shortages they can potentially fix and by the severity of those shortages. For example, if McD is currently suffering shortages of hamburgers *and* fries, then **shuffle** can potentially fix two problems, and so is more relevant than, say, a strategy for correcting service item shortages if only one kind of service item—say, tables—is in short supply.

The **initial-relevance** slot of a strategy frame is used to bias whether McD attends to a situation. **shuffle**'s initial relevance is zero, which means that McD will not attend to it before other strategy frames with higher initial relevance. We have used this mechanism to configure McD to attend to shortages before surpluses and vice-versa.

Once a strategy is selected, its **focus** slot ranks the problems it will deal with. The order of elements in **shuffle**'s focus slot dictates that hamburger shortages should be fixed before fries shortages. This order is determined dynamically, based, in part, on the severity of the shortages.

The **recipe** slot of a strategy frame contains a series of actions—typically a reallocation of resources. For example, **shuffle**'s recipe is a lisp function called `resource-shuffle` that attempts to obtain a resource from somewhere else in the system to assign to the shortage. It has two *tactics* for doing this. One favors obtaining employees who are not assigned to any station. The other looks for employees on stations that are producing surpluses. In either case, if **shuffle** fails to get resources, it returns control to the interpreter.

Sometimes McD's strategy is appropriate but not aggressive enough. For example, a shortage strategy may move one employee to a new station, but two or more employees are needed to correct the shortage in a reasonable time. In such cases, McD may reinvoke that strategy; if **shuffle** moves one employee but is ineffectual, then McD may reinvoke **shuffle** to move another employee.

This raises the question of how we keep track of whether a strategy is having the desired effect. **shuffle** has a **measure-of-progress** slot that contains a function—currently binary—that tells the strategy whether it is

progressing. If the system notices no progress, or if the situation is actually getting worse, the strategy may be reinvoked as described earlier to obtain more resources. McD can measure progress in one of three ways, specified in the **mop-tactics** slot of a strategy frame. **shuffle**'s mop-tactics are `when-improving`, which means that progress is being made so long as the situation (in this case, a shortage) continues to improve. The **measure-of-progress** slot in **shuffle** says that progress is, indeed, being made.

Finally, the strategy has **terminating-conditions** that specify when to quit work. Tactical instantiations of this slot determine exactly when the strategy quits. Three instantiations are

`When-restored`—stay in effect until the problem is solved (e.g., until there is no more shortage).

`When-improved`—stay in effect until the situation improves (even if the problem is not solved).

`Time`—stay in effect for N cycles.

shuffle has a `when-restored` termination tactic, and its **termination** slot contains no, which means that the termination criterion has not been achieved yet.

3.5.1 State

One special frame, called **State**, maintains a global view of McD and is accessible to all strategies. Figure 3.3 shows the state of McD at a particular time during a run (in this case, at `time=8`, as seen in the **system-clock** slot).

State contains some global information that strategies may need to select appropriate tactics. Strategy frames do not maintain global views of McD's world. They are "egocentric" in the sense that they attempt to solve problems as well as possible, irrespective of the global ramifications. For example, a strategy might want to allocate several people to a shortage for as long as necessary to reduce it, irrespective of other shortages. But because strategies compete for resources, a global **State** must tell strategies what they need to know to resolve these conflicts. In particular, strategies consult the **resource-demand** slot (Fig. 3.3), which contains the overall level of demand for resources, when they decide which tactical instantiations to prefer. This slot contains a recommended termination tactic, which may be `restore`, `improve`, or `time`, as discussed above. McD determines which

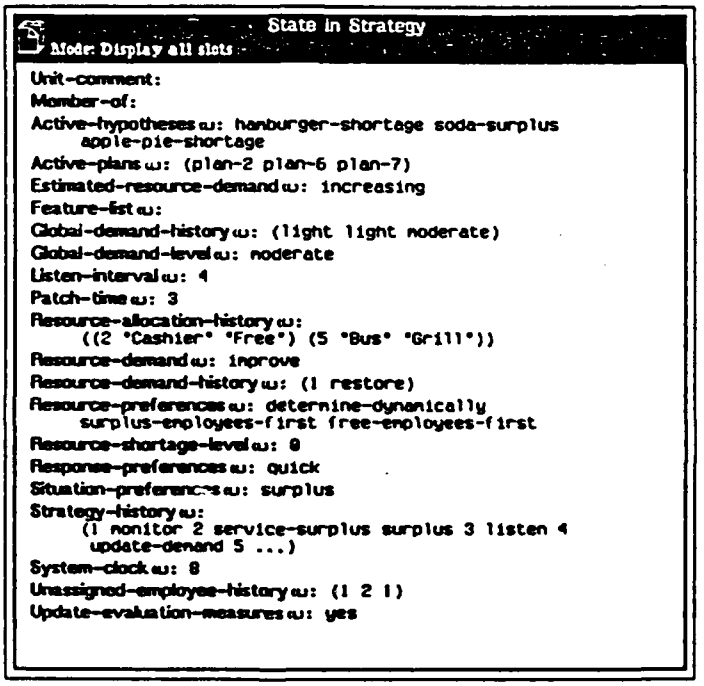


Figure 3.3: The State frame, which represents McD's global state

termination tactic is appropriate by considering known shortages, surpluses, breakdowns, available resources, and rough estimates of trends in the values of these parameters. If the demand for resources is high, then **State** recommends conservative termination tactics—strategies run for N time units and then quit (the **patch-time** slot sets N.) Conversely, if the level is low, then **State** recommends termination tactics that permit strategies to run until they solve their problems. Intermediate levels result in tactics that terminate strategies after some improvement.

State also maintains a list—called **active-hypotheses**—of problems (i.e., surpluses, shortages, and breakdowns) and a list of **active-plans**, which are the problems McD has responded to. The **strategy-history** is simply a list of strategies in the order they are called, so McD can see when strategies were last called. The list in Figure 3.3 shows that in addition to surplus and shortage strategies, McD has strategies to **listen** for incoming orders to **update demand** and other statistics, and to **monitor** measures of progress and termination conditions.

McD maintains a projection of resource demands, called **estimated-resource-demand** that contains a moving average of **resource-demand**. In Figure 3.3, estimated demand is increasing. Obviously, estimated demand can be calculated in many ways; for example, we know that demand for food (and thus resources) increases during breakfast, lunch, and dinner hours, and so might include the time of day in the calculation of estimated resource demands.

The purpose of the other slots in **State** will be described later, when we present examples of McD.

3.5.2 How McD Works

McD has a basic control cycle in which the interpreter first polls strategies to find those that are applicable, then ranks the applicable strategies, and then gives control to the top-ranked strategy. That strategy will attempt to change McD's configuration (i.e., the allocation of resources to stations) and will then return control to the interpreter. If the strategy succeeds—and it may not if resources are unavailable—the new configuration will remain in effect until the strategy terminates it. We now describe these steps in more detail.

3.5.3 Polling and ranking strategies

At each cycle, the interpreter asks each strategy whether it is applicable and thereby discovers whether McD has surpluses, shortages, or breakdowns at any stations. A frame-like structure is created for each problem situation, and a pointer to it is added to the **active-hypotheses** list in **State**. Sometimes, several instances of the same situation arise (e.g., several shortages), and a strategy will be applicable to each. This and other factors are combined by a function in the **relevance** slot of each strategy. Relevance represents the strategy's own assessment of how much it can contribute to keeping McD running smoothly. In general, the more situations to which a strategy applies (and the greater their severity), the higher its relevance score. The strategy with the highest relevance is selected for execution unless another with equal relevance was invoked less recently. Note that, except for this last clause, the selection of strategies is based on information local to the strategy frames. The interpreter then turns control over to the selected strategy.

3.5.4 Executing a strategy

When a strategy gains control, it instantiates itself with *tactics*. The first thing a strategy such as **shuffle** (Fig. 3.2) does is to determine its **focus**, which in this case is a shortage of `hamburgers` and `fries`. Currently, a **shuffle** can select only a single problem (e.g., hamburgers) but eventually it will be able to address multiple problems from **State**'s active hypotheses list—if the problems are on its **focus** list and if **State** permits it the resources. After selecting a problem, **shuffle** selects the most appropriate measure of progress, and also the most appropriate terminating conditions, given the **resource-demand** slot of **State**.

shuffle then tries to run its **recipe**. Since it is an instance of a **menu-item-shortage** strategy, it will try to find additional resources—in this case, an employee to move to a hamburger grill. This is also a tactical issue. As mentioned earlier, the two tactics for finding employees are to favor unassigned people and to favor those on stations producing surpluses. **State** tells strategies which of these tactics to select; the **resource-preferences** slot in Figure 3.3 says to take free employees before those on surplus stations.

3.5.5 When strategies fail

If a strategy like **shuffle** cannot find an employee, it will return control to the interpreter without taking any action, and record its failure in the **resource-shortage-level** slot of **State**. Another kind of failure happens when a shortage plan successfully finds an additional employee, but cannot use him because the station that's producing the shortage is already staffed to capacity. Lastly, a plan can fail if it attempts to take an employee off a station that's producing a surplus, but nobody is working at the station (i.e., the surplus is residual).

3.5.6 Termination of strategies

Assuming **shuffle** succeeds, it will move an employee to a grill and mark that employee as busy. This means that no other plan can grab that employee until **shuffle**'s termination condition is satisfied, at which time the employee is marked as `free`. Recall that when a strategy is executed, it looks at **State** to determine one of three terminating conditions: the strategy is allowed to work until the problem is fixed, or until progress has been made, or for N time units. One tactical issue is whether the terminating conditions of a strategy should be set once, or whether they should be updated on every cycle. The argument for the latter is that if resource levels were very tight initially, but relax over time, then the strategy ought to be allowed additional resources; and, conversely, if resources were plentiful immediately but now are tight, the strategy ought to be allowed fewer resources. Clearly, there are many tactical possibilities for terminating strategies, but currently, McD allows just two:

`static-termination-conditions`—termination conditions are set once when the strategy is created and not changed.

`dynamic-termination-conditions`—termination conditions are updated dynamically on every cycle.

These tactics are selected by the **update-evaluation-measures** slot of **State**.

In special cases, McD needs resources more rapidly than they are provided by the mechanism just described. It then invokes a strategy called the **terminator**. This strategy is always applicable, but its **relevance** is related to the **resource-shortage-level** slot of **State**, so that it is selected for execution only when this level is high. This, recall, is determined by the

number of shortage strategies that fail. Once invoked, the terminator will mark resources as `free` even if their termination conditions have not been met. Three tactics determine how aggressively it does this:

`benign`—find a resource on a station that has just a small problem (i.e., a small surplus or shortage) and that has already made some progress toward solving the problem.

`edgy`—like benign, except the problem can be moderate.

`aggressive`—first look for resources on stations where there has been progress, and mark a resource at the station that has the least serious problem; otherwise mark a resource at the station with the least serious problem; but if all stations have equally serious problems, mark the resource that has been assigned longest.

These tactics are currently selected by the resource-demand slot of State. It takes one of three values, `restore`, `improve`, and `time`, depending on the degree of resource demand (`restore` is the least demand). Resource demand is calculated from historical, current, and anticipated aspects of State.

3.6 Examples

In the following examples, we will show how we tuned the McD system by adjusting the values in slots of strategy frames so it would maintain a fairly "trim" configuration. Although this will not demonstrate the necessity of the slots in question, it will demonstrate their sufficiency. We found that we could quickly improve McD's performance to a point using relatively local information. However, once the system achieved a certain level of performance it became impossible to predict whether further changes would improve or detract from performance, which raises questions about the utility of global information.

We evaluate McD's performance by several metrics. One is a graph, over time, of the supply and demand for various items. Figure 3.4 shows a graph in which McD's performance is poor: Although it doesn't have large shortages or surpluses of food items initially, by time=20 it develops a shortage of hamburgers, then a little later a surplus of fries. Around time=40, McD begins to develop a massive shortage of sodas, and around time=50 a slightly smaller shortage of hamburgers, and then a shortage of

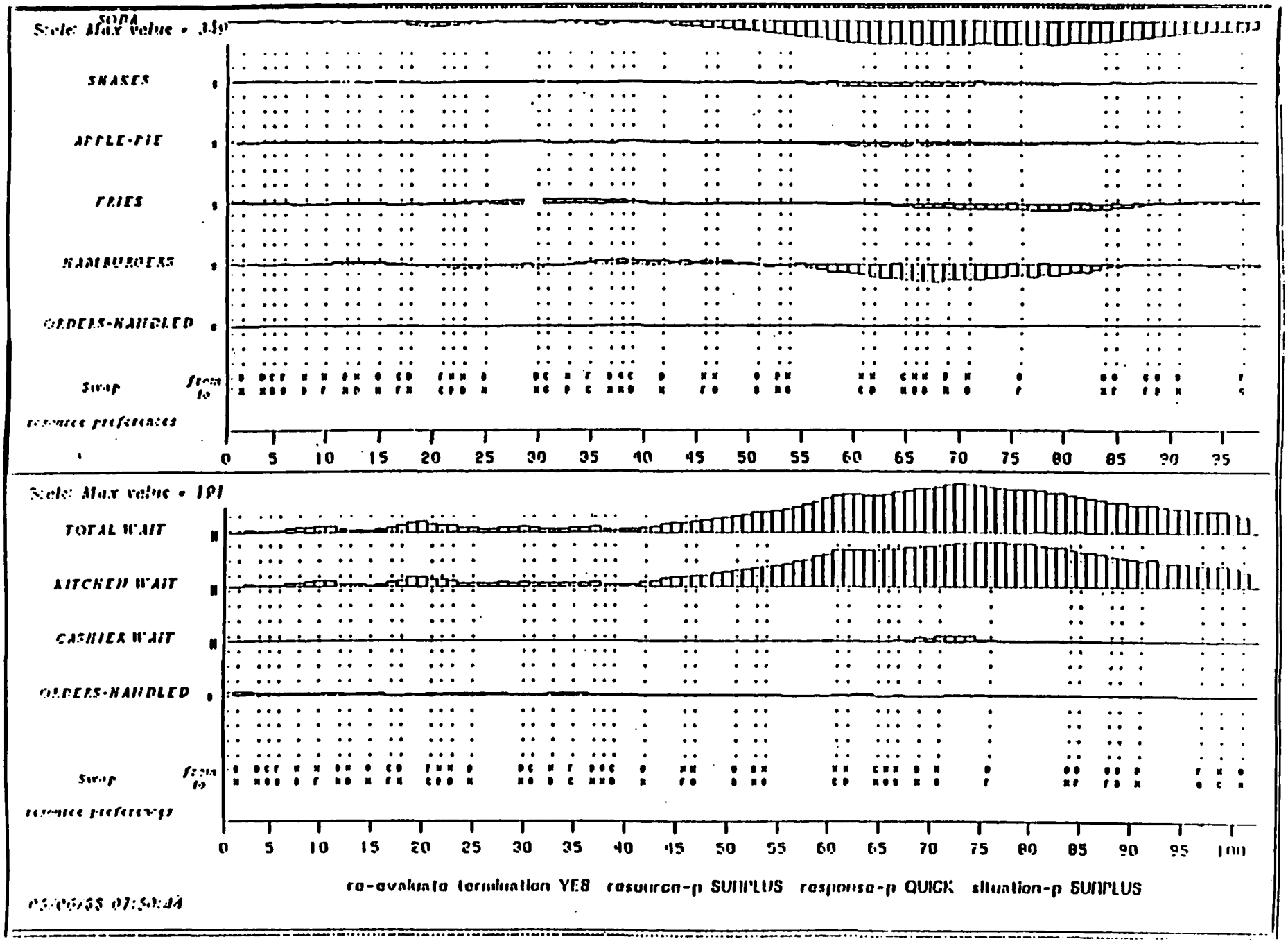


Figure 3.4: Graph of typical performance of MCP

fries. As these shortages develop, one can see a corresponding increase in the length of time that customers must wait. This is due almost entirely to waiting for food from the kitchen, although around time=65 customers must wait a short time for cashiers.

Each movement of an employee is represented in the graph by a vertical dotted line, grounded at a pair of characters that represent the initial and final location of the employee. For example, shortly before time=40, McD takes an employee from a cashier station (C) and moves it to the drinks station (D), probably to correct the slight shortage of sodas. In fact, the shortage disappears around time=40, and shortly thereafter McD moves the employee from the drinks station to the list of unassigned employees (X). This, it turns out, was a bad move, since the drinks shortage immediately becomes very serious.

These moves are proposed by strategy frames based on relatively local information: A frame sees a shortage, surplus, or breakdown, and proposes itself to the interpreter. The choice by the interpreter among strategy frames is also based on relatively local information: It selects the strategy frame that is most relevant, that is, the one that potentially fixes the most problems of greatest severity. It does not predict the effects of the resource allocations suggested by each relevant strategy, and in particular does not predict the interactions between these resource allocations and the ones that are already in effect.

McD's performance on each run is summarized with a table of statistics (e.g., Figure 3.5 presents the statistics that correspond with Fig. 3.4). We record the mean surplus, shortage, and total supply of each food item, the standard deviation, and the number of each situation. In Figure 3.5, there are 33 situations in which we had a surplus of hamburgers, in which the mean surplus was 30.18 with a standard deviation of 17.98. More significant, of course, is the 65 shortage situations, in which the mean shortage was a distressing -86.31! Overall, the mean level of supply for all 98 cases (including shortages and surpluses) was -47.08. We have a similar pattern of results for the other food items. A summary of these figures is found in the row called "Totals," which contains the weighted means of surpluses, shortages, and overall supply for all the food items. The weighting reflects the rate at which food items can be produced and are typically consumed. Thus, big shortages of hamburgers "count the same" as smaller shortages of apple pies, because the latter are produced more slowly. Figure 3.5 also includes the average wait for a cashier, and for food once an order has been placed; these figures are 0.27 and 12.20, respectively, indicating that a cus-

tomer can expect to order in about one-quarter of a time unit, but wait 12 time units for food! Lastly, Figure 3.5 includes information about the efficiency with which McD uses its employees. We record the total number of units worked by all employees and the number of those units in which they were productive (not idle). The ratio, or utility mean, is just the ratio of these statistics. In this case, the employees were not particularly efficient (71%).

McD's performance in Figure 3.4 and Figure 3.5 is pretty poor. Now we will show how, by modifying the slot values of McD's strategy frames, we improved performance. The previous example and all the subsequent ones use the order set shown in Figure 3.6. It contains 100 *blocks* of orders. Each block contains between zero and eight orders (distributed around a mean of four orders), and each order includes zero to seven orders of pies, burgers, fries, sodas, and shakes.

The performance illustrated in Figure 3.4 and Figure 3.5 is poor for several reasons. First, the mean wait for food is over 12 units—much too long. This is due in part to the enormous mean shortage (-53.9): McD is out of everything most of the time, so customers have to wait. Indeed, there are 308 shortage situations and 176 surplus situations. Our immediate goals were to reduce these shortages and reduce the waiting time for food.

By changing the values of slots in strategy frames, we were able to affect the following aspects of McD's behavior:

Preference among situations. McD can react more quickly to surpluses, or more quickly to shortages, or equally to both, as determined by the **situation-preferences** slot of **State**.

Where to get resources. McD can either favor taking resources from stations that are producing a surplus, or it can favor taking the resources from a list of free resources. In the first case, if no stations are producing surpluses, McD will look to the free list; in the second case, conversely, if there are no free employees, McD will look at stations producing surpluses. This choice is dictated by the **resource-preferences** slot of **State**.

Speed of response. McD can react quickly or slowly to situations, as dictated by the **response-preferences** slot of **State**.

Whether to re-evaluate termination conditions. McD can allow strategy frames to re-evaluate their terminating conditions dynami-

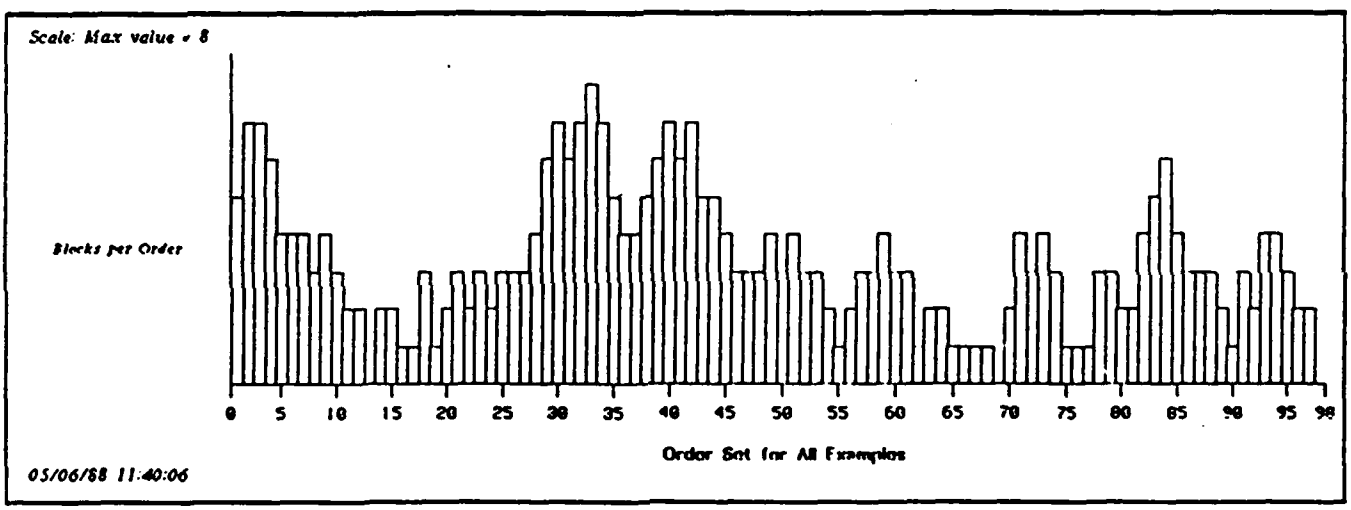


Figure 3.5: Statistics for Production and Utilization from the Initial

re-evaluate termination YES resource-p SURPLUS response-p QUICK situation-p SURPLUS

		Surplus	Shortage	Total
HAMBURGERS				
	mean	30.18	-86.31	-47.08
	st. dev	17.98	75.92	83.44
	number	33.00	65.00	98.00
FRIES				
	mean	25.94	-41.66	-8.82
	st. dev	17.28	30.73	41.90
	number	47.00	50.00	98.00
APPLE-PIE				
	mean	9.17	-19.63	-5.91
	st. dev	4.95	14.99	18.26
	number	46.00	51.00	98.00
SHAKES				
	mean	7.80	-27.55	-11.63
	st. dev	4.71	20.78	23.31
	number	41.00	53.00	98.00
SODA				
	mean	19.11	-154.92	-138.94
	st. dev	9.96	120.10	125.04
	number	9.00	89.00	98.00
Totals (weighted)				
	mean	17.11	-53.91	-27.74
	st. dev	11.94	52.92	54.53
	number	176.00	308.00	490.00

Avg. Wait over all Customers:		St. Deviation
Cashier Wait	0.27	1.37
Kitchen Wait	12.20	17.95

Employee Stats:

Total Productive time units	478
Total On Duty time units	678
Utility mean	0.71
Utility st. deviation	0.19

Figure 3.6: The order set used for all examples

cally, or it can set terminating conditions once for each strategy as it is invoked. This is determined by the **update-evaluation-measures** slot of **State**.

The version of McD discussed earlier was configured to respond to surpluses before shortages, to get employees from surplus stations before free employees, to respond quickly, and to re-evaluate its terminating conditions. Since this version generates huge shortages, it makes little sense to attend to surpluses before shortages. Thus we ran the system again with it attending to shortages first. The results were much better: The mean surplus was 24.7, up from 17.1; but the mean shortage was -32.2, down considerably from -53.9; and the total mean supply was -4.9, down from -27.4. Moreover, the cashier wait dropped from 0.27 to 0.08; and more importantly, the kitchen wait dropped from 12.2 to 6.5. There was no improvement in the efficiency with which McD used its resources. Clearly, attending to shortages before surpluses improves McD's performance.

Still, it's awkward to wait 6.5 time units for one's food. We also noted that cashier wait was almost nonexistent. If cashier wait was increased slightly, then it might delay some orders being placed and reduce the shortages, and thus the kitchen wait time. In our next run we changed the applicability conditions for cashier shortages and surpluses, to register a shortage less quickly and to register a surplus more quickly. (Although these functions are not shown in any of the Figures in this paper, they too are explicit and easy to modify.) The net result, we hoped, would be to free up cashiers and make them easier to reassign. At the same time, we changed the **resource-preferences** slot in **State** so McD would look for resources on the free list before surplus stations. The net result was to improve performance slightly. The cashier wait increased slightly (from 0.08 to 0.42) and the kitchen wait decreased slightly (from 6.5 to 5.8). At the same time the mean surplus decreased a little and the mean shortage improved a little.

Beyond this, we had little success improving McD's performance. This is a story that has repeated itself many times: We can get performance to some reasonable level by a small number of configuration changes, all of which rely on relatively local information. After that, improvements in performance seem to require relatively global information that is difficult and sometimes impossible to acquire. For example, we tried changing the applicability conditions for cashier shortages, above, in the hope that, by making it harder to get to a cashier, fewer orders would be placed, and so the wait for food would be decreased. This argument, based on non-

local interactions of assignments of resources, suggests that relatively global information could improve McD's performance; but as noted above, this strategy had only a small effect. In fact, most of our attempts to predict the effects of interactions of resources, and so improve overall performance, failed. Beyond the effects of the really big changes, which can be selected based on local information, the other effects are too susceptible to the global interactions of many variables, and levels of shortages and surpluses, and other dynamic factors, for us to predict them.

3.7 Conclusion

Starting from the position that complex control strategies can be viewed as the interaction of smaller strategic units, we developed a representation for these units and showed how their tactical instantiations could be adjusted to tune control strategies. We showed that particular slots of strategy frames are sufficient to control McD in a dynamic environment, but we did not show their necessity. Nor were we able to predict the effects of changes to all slots. Clearly then, research with strategy frames is nascent.

Future research with strategy frames will emphasize the subtle interactions between how strategy frames are instantiated with tactics and the dynamic environment in which they are used. That is, we want better predictability. This is essential if we are to study the tradeoffs between the costs and benefits of relatively local and global information. Currently, we have no reliable methods to predict the global interactions of strategies' allocations of resources, so we can't guess at the utility of this knowledge. This is our priority for work in the future.

Strategy frames were motivated by the need for declarative representations of strategy, and by the observation that, in many systems, devices with similar functionality had been implemented in an ad hoc way. Ideally, strategy frames will become part of the AI programmer's toolbox, complete with tactical instantiations for various tasks, and interpreters capable of dynamically selecting the most appropriate instantiations. Before that, there is much work to be done: Strategy frames imply "emergent" control, that is, control strategies that arise from interactions of individual control decisions. In McD and in general, these decisions allocate computational and other resources. The basic question is whether the global interactions between these decisions must be examined before they are taken, or whether acceptable control can be based on relatively local information. Our preliminary

experiments suggest that local information is sufficient to replace grossly inefficient configurations of McD with more efficient ones, but beyond that, performance depends on currently unpredictable, global interactions among resource allocations. Further progress depends on making these interactions more predictable.

Part III

Complex and Dynamic Environments

Chapter 4

The Centrality of Autonomous Agents in Theories of Action Under Uncertainty

4.1 Introduction

Reasoning under uncertainty has two aspects. One is to assess the most likely states of the world, the other is to act on t[illegible]e assessments. The former is often called *judgment* and the latter *decision making*. Judgment is the primary focus of research on reasoning under uncertainty in artificial intelligence (AI). Although many AI systems make decisions, those that serve as examples of research on uncertainty typically do not. Instead, the literature on uncertainty in AI is concerned almost exclusively with a single aspect of a single generic task: combining evidence in interpretation tasks. Expert systems for medical diagnosis serve as the canonical AI systems in this research. Other aspects of interpretation tasks that *do* involve decisions, such as deciding which evidence to acquire and deciding how to treat a patient given a diagnosis, have been largely ignored. Other generic tasks and domains, such as planning, design, robotics, and process control, have also been neglected. But AI is increasingly concerned with decision-making in such tasks and domains, and is much less concerned with judgment in simple diagnosis. Thus, the research published in, say, the proceedings of the AAAI Workshops on Uncertainty and Artificial Intelligence is largely irrelevant to

many current AI research problems, even though these problems involve considerable uncertainty.[1]

This paper describes a class of problems that concern how agents act in uncertain environments. Lately, these have been called *planning problems* and the agents *planners* (e.g., [McDermott, 1987], [DAR, 1987]). We are writing this paper to urge researchers in the uncertainty community to expand their focus to include planning problems. We advocate this for several reasons. First, although uncertainty is the most salient characteristic of planning problems, current research in the uncertainty community tells us nothing about how to design planners. Second, much of the literature on planning comes from the logicist community in AI (e.g., [Georgeff and Lansky, 1987]), and so emphasizes nonmonotonic reasoning over probabilistic approaches to uncertainty. But probabilities (if you can get them) are better suited than assumptions to the task of selecting actions, because they can be combined with utilities (if you can get *them*) and ranked.

Third, we believe that too much energy is devoted to making ever-finer distinctions between methods for combining evidence; just as symmetric efforts are devoted to showing that these methods in fact mathematically subsume one another, or can be incorporated one within the other. These debates ignore a fundamental question: for what *purpose* are we combining evidence? Interpretation tasks provide the illusion of a purpose; but when a researcher from the uncertainty community says "medical diagnosis," he or she usually means "evidence combination" only, not planning the diagnosis, not deciding between treatments, and not prognosis. In contrast, planning depends on evidence combination and so provides a context for research on judgment. Planners need to interpret data from the world *well enough* to act; they need to know the likely outcomes of actions well enough to select

[1] For example, of 51 papers published in the Proceedings of the Third Annual Workshop on Uncertainty in Artificial Intelligence, roughly one-half mentioned no application whatsoever but were clearly influenced by diagnostic expert systems, two described vision systems, and all but four of the rest described diagnostic applications, or algorithms for learning or knowledge acquisition for diagnostic applications. The remaining four are Tong and Appelbaum's discussion of the relationship between knowledge representation and evidential reasoning in information retrieval tasks [Tong and Appelbaum, 1987]; Steve Hanks' discussion of the persistence problem in planning (i.e., how does the passage of time affect one's belief in propositions) [Hanks, 1987]; Cohen's description of a program and an architecture for planning diagnoses [Cohen, 1987b] (see also Secs. 4.4.1, 4.4.2 of this paper); and Wellman and Heckerman's analysis of the range of tasks facing all intelligent agents in moderately complex environments [Wellman, 1987].

among the actions; they need to know the probabilities of events beyond their control well enough to prepare for them. How well is "well enough"? Let us not debate this in abstract interpretation tasks, but in the context of planning tasks.

Planning is concerned with the interaction of agents and their environments. Time is an important dimension of this interaction: agents may act or remain inactive, but time still passes. Agents are assumed to have goals, among them, finding out about the environment, changing it in some way, changing themselves, changing their relationships with each other and with the environment, and so on. Agents also have plans, which for now are just internal structures that dictate how agents respond to their environment. Plans may be reflexes, multi-action schemas, symbolic contingency plans, and so on. Agents sense their environments. Often, many layers of inference separate sensation from perception. Agents also adapt to their environments.

Uncertainty is the most salient characteristic of the relationship between agents and their environments. Consider some sources of uncertainty: Agents' knowledge about the environment may be inaccurate and incomplete. The space of possible futures expands combinatorially, so agents cannot foresee the outcomes of more than a few actions. An agent is typically not the only actors in environments, and the behavior of other agents may be unpredictable. The environment itself may be unpredictable. Agents may have limited time to reason and act; to meet deadlines they may have to rely on heuristic, approximate, and thus uncertain methods [Lesser *et al.*, 1988]. Time also introduces questions about persistence; for example, how does passing time affect belief in a predicate such as "can't eat another thing" [Hanks, 1987]?

Time is important for another reason: agents themselves persist and may have many opportunities to achieve their goals, so the "one shot" view of decision-making is typically not appropriate. Wellman and Heckerman [Wellman, 1987], who introduce the term one-shot, point out that in most situations, agents are not *required* to decide anything, but can instead collect data, converse, run experiments, and so on. Decisions are rarely unrecoverable: agents can usually recover from decisions with bad outcomes at some cost.

Many tasks can be described in terms of agents interacting with their environments. Those studied in AI include robot path planning, process control, intensive care unit monitoring, learning to avoid air traffic control mishaps, planning diagnoses, and fighting forest fires. All these tasks de-

mand judgment, that is, combining evidence to assess the current state of the world. But they also require decisions about how to act in pursuit of goals, in environments that are uncertain for the reasons described above. We urge the uncertainty community to embrace these tasks because they offer much richer opportunities to study reasoning under uncertainty—especially decision-making—than simple interpretation tasks.

The following sections discuss planning from three perspectives. Section 4.2 focuses on the design of planners, Section 4.3 presents an overview of the AI planning literature in terms of these design issues, and Section 4.4 discusses planners we have built specifically to study reasoning under uncertainty.

4.2 Selecting Actions Under Uncertainty

How do agents select actions to achieve their goals in uncertain environments? AI and related fields offer a somewhat bewildering array of answers. Some are very general; for example, best-first search [Barr and Feigenbaum, 1981] and decision analysis [Winterfeldt and Edwards, 1986] dictate that agents should "do what's best," as assessed by unspecified heuristic evaluation functions and utility functions, respectively. General planning algorithms find sequences of actions, prior to their execution, to satisfy goals and constraints (e.g., [Cohen and Feigenbaum, 1982]). In fact, this is a traditional view of planning; today, planning denotes several other methods, most of which assume some knowledge about the environment. These include case-based planning, where agents recall, modify, and execute plans from memory [Hammond, 1986], [Sycara, 1987]; and reactive planning, where agents are constructed to respond automatically to changes in the environment (see Secs. 4.3, 4.4.1 for details). Finally, we have seen many *control* approaches to selecting actions under uncertainty. Control strategies specify how programs should act. New control strategies are invented when programs do not behave properly (e.g., they ask questions in the wrong order, or consider alternatives irrespective of their prior probabilities). Sometimes, we can pry apart control strategies from their implementations, but in general this is difficult [Gruber and Cohen, 1987a]. We have discussed the problem of selecting actions under uncertainty from the perspective of control in recent papers [Cohen, 1987b], [Cohen *et al.*, 1987a], [Cohen *et al.*, 1987b], and surveyed the relevant literature in [Cohen, 1987a]. Here, we do not

discuss control except in Section 4.4.2.

In this section, we describe the issues that arise when designing autonomous agents that operate in complex, real-time, dynamic environments. These issues drive our work on reasoning under uncertainty.

4.2.1 Internal and external action

Are the actions taken by the agent *internal* or *external*? Intuitively, this is the distinction between thinking and acting. For example, chess requires an internal search of moves and countermoves before committing to an external action, an irrevocable move. Similarly, planning traditionally involves searching internally for an ordered sequence of actions that will achieve a goal, then executing them externally. Many AI systems take no external actions at all; for example, some natural language parsers simply take input and process it internally. Other systems' external actions are limited to requesting data; for example, the MUM system (see Sec. 4.4.1) is designed to ask medical questions in an appropriate order.

4.2.2 Balancing internal and external actions

What factors affect the balance between internal and external action? Should we explore the space of alternative actions before committing to one, or should we commit to the first action that seems appropriate? The answer in general depends on the costs and benefits of searching through the outcomes of external actions. This search is referred to in the planning literature as *projection*. Sometimes we cannot project, or projected outcomes may be irrelevant, or we may lack the resources to compute them. When projection is uninformative, we may simply execute an action to see what happens. A related question is: when should a problem-solver anticipate the outcomes of *multiple* external actions? Since actions are typically not independent, the cost of searching the space of joint outcomes is combinatorial in the number of actions. If, in addition, we cannot predict some of the outcomes, then searching this space may not be worth the effort.

4.2.3 Representing the external world

How accurate is the internal representation of the external environment? We assume that an AI program's internal representation of a chessboard is accurate, at least with respect to the positions of pieces. Traditional planners made similar assumptions: the environment is precisely as we represent it

and doesn't change except by our action, and our knowledge of these changes is complete and accurate. Clearly, planning in the real world cannot proceed on these assumptions. Much of the discussion later in this paper is about how to plan when the internal representation of the external environment is incomplete and inaccurate.

Uncertainty in a planner's world model comes from several sources, including these:

- The planner may have an inaccurate or incomplete understanding of the principles that govern the dynamic behavior of the environment.
- The planner may have inaccurate information about the current state of the environment.
- The system may not have adequate time (or other resources) to assess the state of the environment, the outcomes of actions, or changes in the environment beyond its control.

4.2.4 Real-time constraints

Are real-time constraints placed on the agent by the environment? As agents are designed to have more autonomy, we must be concerned about the timeliness of their planning, plans, and actions. We regard real-time planning problems as falling between the extremes of adequate time to produce a good solution and inadequate time to produce *any* solution. Real-time planning is thus concerned with tradeoffs between the quality—broadly construed—and the time requirements of a plan. We do not regard faster computers or more efficient planning algorithms as solutions to the real-time planning problem in general, though they will obviously help in specific applications. The general real-time problem starts with the premise that the available time will at some point be inadequate. It demands that our planners adapt their processing to produce the best possible solution in the available time. This has been called *approximate processing* because it implies that the solution to a problem will approach but not meet all of our goals [Lesser *et al.*, 1988].

4.2.5 Constraints between actions

Actions are rarely independent; can dependencies between actions help select actions? Some actions interfere, so that one prevents another or undoes the outcome of another. Some actions are redundant, so one achieves the same state or gets the same evidence as another. If the outcomes of actions

were certain, then redundancy might be inefficient; but we can also exploit redundancy to make up for uncertainty about the outcomes of actions. In addition, constraints between actions will determine the extent to which taking one action commits the agent to a particular series of actions in the future, and so emphasizes the importance of considering such constraints before premature committment to a single action.

4.2.6 Where do plans and goals come from?

Do actions come from internally-generated goals and plans, or are goals and plans epiphenomena of the direct interaction of the agent with its environment? In most AI planner, goals and plans select actions. But the environment can play the same role. Thus, an agent that has no goals or plans but responds to its immediate environment will appear to have goals and plans, if the environment provides regularity and continuity. From this perspective, goals and plans are not explicit structures within the agent but emerge from its interaction with its environment. An intermediate position is associated with "Simon's Ant" [Simon, 1981]: the problem solver is assumed to have goals and plans that bias its selection of actions, but the environment also plays a greater or lesser role.[2]

4.2.7 Global and local evaluation of actions

What is the relationship between global properties of plans and local planning decisions? Plans are generated by selecting actions, so to generate "good" plans one must somehow evaluate the potential component actions. It is prohibitively expensive to calculate the extended ramifications of actions, so many planners base decisions about actions on their local outcomes. But sequences of actions that look good from a local perspective are not necessarily good plans, because they may not satisfy global criteria; for example, an agent that always selects the cheapest applicable action will not find the cheapest plan if that plan requires a sequence of actions ordered by the inverse of their costs. The agent's evaluation of the local outcome of actions, which may frequently ensure low-cost plans, does not guarantee the lowest-cost plan in all cases.

Real-world plans must satisfy so many criteria that agents will not be able to generate optimal plans, that is, plans in which the outcomes of all

[2]Simon notes that the path of an ant in a sand dune appears on a fine scale to be almost random, since the ant must respond to random obstacles. Yet it has a general direction dictated by where the ant wants to go.

actions look as good as possible from a global perspective. Plans should be inexpensive, flexible, and likely to succeed; they should take relatively little time to generate, require little monitoring, and so on. Since optimality is not a realistic aim, it is neither realistic nor desirable to project the global ramifications of actions. Nevertheless, agents can get into serious trouble if they don't project to some depth the ramifications of some actions. In Section 4.4.2, we discuss the characteristics of the environment that require agents to project; and conversely, the kinds of environments in which agents can generate good (if not optimal) plans without any projection, that is, by evaluating only the local ramifications of outcomes of actions.

4.2.8 Adaptation

A problem solving agent must have knowledge to act in the ways advocated in this paper. As we note in Section 4.3, general planners are weak, but powerful planners require knowledge about their environments. This raises three issues: what is learned, how is it learned, and what form does it take? To plan by projection, agents must know which actions are applicable and which outcomes are possible, given their goals and the state of the environment; and they must evaluate those outcomes by projection. Planning without projection also requires knowledge about the applicability of actions, but it may not require anything else. An agent may simply maintain a list of situation-action pairs that tell it what to do in each state of the environment. In this case, the agent's actions are selected by its environment. Such agents are *adapted* to their environments to the extent that they have learned which actions are appropriate in which states of the environment. We believe that most agents will project in some situations and simply react in others, so both kinds of knowledge must be learned.

One advantage of reaction, as opposed to projection, is that agents can learn how to act in the absence of explicit knowledge about the dynamics of the environment. For example, one can learn to ride a bicycle—that is, how to balance, steer, and so on—without knowing the physics of balancing and propelling the bicycle. Anecdotal and scientific evidence indicates that one should *not* ride a bicycle by projecting the ramifications of each tilt and wobble. The reason we do not need to project in these situations is that their outcomes always depend on predictable interactions of the same factors—gravity, one's angle, velocity, and so on.[3] Action sequences such

[3]Paradoxically, projection should be reserved for situations where the relationships between actions and outcomes are not completely predictable; if they *were*, then they

as pedalling a bicycle are often called skills. In humans they tend to be automatic, that is, to require no conscious effort.[4] People work hard to develop skills in real-time environments such as driving cars, flying jets, and most sports, because these environments do not afford the opportunity to think (see Sec. 4.2.4).

Although most knowledge bases are built by hand, agents in complex environments will have to learn autonomously how to act. Even in a simple medical domain, we found it necessary to augment standard knowledge-acquisition mechanisms with an ability to automatically generalize the acquired knowledge to a broader range of situations (see Sec. 4.4.2). AI has developed many learning mechanisms (e.g., [Michalski *et al.*, 1986]) that may be appropriate for agents learning how to act.

The knowledge that agents need to select actions can take many forms, including situation-action rules, general preference rules (see Sec. 4.4.2), contingency plans (see Sec. 4.4.2), utility functions, scripts, and so on. The form of the knowledge depends on how it will be used, and in turn on the demands placed on the agent by the environment. For example, we noted above that humans adopt automatic skills in time-constrained environments; unfortunately, skills are inflexible and difficult to interrupt. Thus, agents must learn skills that run to completion before the environment changes in ways that make them inapplicable.

4.3 An overview of the planning literature

AI research in planning, which has concerned itself more than any other part of AI with the problems of selecting actions to achieve goals, can be viewed from the perspective of these dimensions. Early on, researchers adopted predicate calculus as a representation language for planners and viewed planning metaphorically (and sometimes literally) as theorem proving. The initial state of the world was captured in a set of axioms, the goal state was a theorem, and the plan was a proof that the goal state could be reached from the initial state. This approach to planning is found in HACKER [Sussman, 1975], GPS [Newell and Simon, 1972], STRIPS [Fikes *et al.*, 1972], INTERPLAN [Tate, 1974], Waldinger's planner [Waldinger, 1977], ABSTRIPS

would eventually be learned as situation-action pairs.

[4] For a discussion of the distinction between automatic and controlled behavior, see [Schneider and Shiffrin, 1984]. Computational models of these behaviors and their interactions have been sugggested by Schneider [Schneider, 1985] and Day [Day, 1987b], [Day, 1987a].

[Sacerdoti, 1974], NOAH [Sacerdoti, 1975] and NONLIN [Tate, 1977].

From our perspective, the most salient characteristic of these planners was their almost complete avoidance of uncertainty (see Sec. 4.2.3). These programs were crafted under the assumptions that they would have complete, accurate knowledge about the state of the environment, and complete, accurate knowledge about the *immediate* outcomes of all actions. Actions were represented (and indexed) in terms of their immediate outcomes; for example, in the blocks world, one of the immediate outcomes of the action *(take-off x y)* is *(clear-top y)*—removing x from y clears off the top of y. But although these planners knew the immediate outcomes of actions, they were required to search combinatorial spaces of extended ramifications. For example, an action such as *(put-on red-block green-block)* may achieve an immediate goal, but may later impede progress toward a goal that requires *green-block* to have a clear top. A better plan may involve moving *green-block* first and then putting *red-block* on it. Early planners dealt with uncertainty about the extended outcomes of actions by various forms of search. Some algorithms were more efficient than others (i.e., required less backtracking). But all assumed that, because the immediate outcomes of actions are certain, the extended ramifications could be discovered by projection, that is, by internally simulating the actions in a plan before committing to any external actions.

Because planners were assumed to know the current state of the environment and the outcomes of all actions, and because it was assumed that the environment did not change except through the actions of the planner, there was really no need to distinguish internal and external actions. Actions could not introduce discrepancies between the projected representation of the world and the actual world.

More recently, AI has developed planning methods that do not depend so heavily on these assumptions. One approach, which modifies the earlier planning algorithms relatively little, involves *replanning* when the environment turns out to be different than projected. For example, Wilkins' SIPE planner was very much like NOAH [Sacerdoti, 1975], but when discrepancies were detected between the environment and its internal representation, SIPE would efficiently modify its plan, maintaining as much of its original plan as possible [Wilkins, 1985]. Related replanning methods have been developed by Broverman and Croft [Broverman and Croft, 1987].

Early planners assumed that actions were instantaneous, and that their effects persisted until they were explicitly negated. However, actions take time, and the states they bring about may be ephemeral. Temporal logics

and temporal planners address these issues (e.g., see [McDermott, 1982], [Allen, 1984], [Dean, 1987a] for work on temporal logic, and [Vere, 1981], [Hanks, 1987] for temporal planners).

Just as actions take time in real physical environments, planning and replanning themselves take time (see Sec. 4.2.4). In real-time planning, the agent must allocate a limited resource—time—to planning, replanning, monitoring the environment, and action. Time usually has costs (e.g., in the fire-fighting domain discussed in Sec. 4.4.3, forests are consumed while the agent plans.) The balance between internal and external actions is further complicated by uncertainty about the environment: to meet time constraints, agents may begin sequences of actions before they have all the evidence they need; but if they wait, an opportunity may be lost and the evidence will be of no value. Recent work on real-time planning includes that by Durfee [Durfee, 1987], Lesser, Durfee and Pavlin [Lesser *et al.*, 1988], Hayes-Roth and her colleagues [Hayes-Roth *et al.*, 1986], [Garvey *et al.*, 1987], [Pardee and Hayes-Roth, 1987], Korf [Korf, 1987], Dacus [Dacus, 1987], Luhrs, et al. [Luhrs and Nowicki, 1987], Herman, et al. [Herman and Albus, 1987], Daily, et al. [Daily *et al.*, 1987], Firby and Hanks [Firby and Hanks, 1987], Hendler and Sanborne [Hendler and Sanborn, 1987] and others.

A radically different approach challenges the distinction between planning and execution, and thus the distinction between internal and external action (see Secs. 4.2.2, 4.2.6, 4.2.7). This view holds that, by any metric, projection in uncertain environments is inefficient: dynamic environments will never be as they are projected to be, so projection is a waste of resources. Projection involves selecting actions that are expected to be appropriate at some point in the future; planning without projection involves selecting actions based on the current, immediate environment, without explicitly considering their consequences. ***Reactive planners*** do not project, but simply react to their environments, so the distinction between planning and execution is absent [Chapman and Agre, 1987], [Agre and Chapman, 1987], [Brooks, 1985], [Firby, 1987].

Reactive planning raises questions about the status of goals: planners may *appear* to be goal-directed when, in fact, they are simply responding to their environments (see Sec. 4.2.6). One does not need internal structures called goals to explain apparently intentional behavior. But we believe that intelligent agents should reason about their goals, so some goal-directed behavior will not be generated by reactive planning [McDermott, 1987], [Dean, 1987b].

Reactive planners need to know *how* to respond to different situations. They recognize situations and respond appropriately, so to be adapted to complex environments, reactive planners need to recognize many situations. This requires so much knowledge that it will be impossible to build reactive planners for some environments; instead, reactive planners must learn how to respond to situations through interactions with their environments (see Sec. 4.2.8. Some AI techniques for learning concepts have been suggested or applied to the task of learning the situation-action contingencies required for reactive planning; these include connectionist learning [Barto *et al.*, 1983], [Sutton and Barto, 1981], [Jordan, 1986,Rumelhart and Norman, 1982], knowledge acquisition and generalization [Gruber, 1987] (see Sec. 4.4.2), chunking [Laird *et al.*, 1986], and production rule learning [Mitchell *et al.*, 1983], [Anderson, 1983]).

We believe reactive planning is an extreme response to uncertainty in the environment. We agree that the value of projection depends on the certainty with which we can predict the outcomes of actions; but since this is variable, and depends on many factors, we do not agree that planners should completely forego projection. (Others have made similar observations [Hayes-Roth, 1987], [Dean, 1987b], [Swartout, 1987].) We illustrate this later in the context of two of our own planning systems (Sec. 4.4).

We can, in fact, project even if we do not know the precise outcomes of actions. Decision analysis is a form of planning by projection: When actions have uncertain outcomes, and these lead to further actions, then a combinatorial space of actions and outcomes is quickly generated (see Sec. 4.2.3). If one knows the probabilities that actions will lead to particular outcomes, and also the utilities of the outcomes, then one can find the ***subjective expected utility*** of actions. For example, imagine test-1 costs $10 and will accurately say whether or not a patient has disease A, but says nothing about diseases B and C; and test-2 costs $50 and is diagnostic for disease B but says nothing about A or C. Which diagnostic action should we take first, test-1 or test-2? Assuming A, B, and C have equal priors, it is most efficient to do test-1 first. Decision analysis will resolve the question for the general case of unequal priors.[5]

But note that two assumptions are implicit in the example: the statement of the problem implies that the hypotheses A, B, and C are both ***mutually exclusive*** and ***exhaustive***. Thus, if test-1 finds A, we need not do test-2; and if both tests fail to find A and B, then the answer must be C.

[5] We are grateful to Professor Glenn Shafer for this example.

Mutual exclusivity is a special case of conditional probability: When we say A and B are mutually exclusive we mean that the probability of disease B is conditional on the outcome of test-1 (and equivalently, our belief in A), and vice-versa. Thus, in the general case, to plan a sequence of tests to find out which disease a patient has, we need to know the conditional probability of each disease given each combination of outcomes of tests for these diseases. Even if we assume the diseases are mutually exclusive and exhaustive, and we assume that every test either confirms or disconfirms a disease (but does not provide partial support for any disease), we are still faced with a combinatorial search because there are N! sequences of diagnostic actions, and because each action can have several possible outcomes.

We are not rejecting decision analysis as a technique for planning under uncertainty, only noting that its inherent combinatorics must be managed carefully (e.g., Wellman has proposed some approximate forms of decision analysis [Wellman and Heckerman, 1987]). All planning by projection generates combinatorial spaces of plans; uncertainty about outcomes simply makes the problem worse. We expect that decision analysis can be merged with AI planning techniques (e.g., least commitment and hierarchical planning) to reduce the combinatorics of projection. For example, hierarchical planning reduces the combinatorics of projection by generating plans at successive levels of abstraction [Cohen and Feigenbaum, 1982]. Decision analysis might be used to select actions at each level.

4.4 Case studies in planning under uncertainty

In this section we describe some of our research on planning under uncertainty. We begin with MUM, a system for planning medical workups. MUM led to a shell for building planners called MU, which in turn is the basis of a knowledge-acquisition system called ASK. We also describe a system called PLASTYC which, though not yet completed, highlights differences between reactive planning and planning with projection, and also illustrates how simulators can facilitate research on planning under uncertainty.

4.4.1 MUM

The MUM system plans diagnostic workups of chest pain [Cohen *et al.*, 1987a]. Its goal is to ask questions, request tests, and prescribe therapies in an efficient order. By efficient, we mean that MUM should not take a sequence of actions (a diagnostic plan, or

workup) to gain evidence if another sequence would be as informative but less expensive. Viewing this as a traditional planning problem, we would project the outcomes of all sequences of evidence-gathering actions and select the sequence that provides us with maximum diagnosticity for minimum cost. Planning diagnoses is then a matter of searching this space.

In MUM we assume that the space of diagnostic plans or workups is too large to search exhaustively; that is, we will be unable to generate the most efficient workup. We propose instead that workups are generated one action at a time, or equivalently, that the search for diagnostic plans proceeds incrementally, with each action being executed before the next is contemplated (see Sec. 4.2.7). MUM is essentially a reactive planner because it's actions are determined largely by its environment (the state of the patient) and its preferences. The search is guided by heuristics that we call *preferences*. One preference is to ask cheap questions first; another is to ask diagnostic questions first. A more specific preference resolves the conflict that can arise between these two: if the patient is in danger and the most diagnostic action is also the most expensive, then take that action; but if the patient is not in danger, take the cheaper action. In the worst case, we might require a specific preference for every situation that could arise, but in practice we can generate moderately efficient workups with relatively few preferences.

MUM has two components: an interpreter, and an inference network with nodes representing evidence-gathering actions at the bottom, intermediate conclusions in the middle, and diseases at the top. MUM's interpreter selects and executes an evidence-gathering action and propagates the data it acquires through the inference network; then, on the basis of its preferences and the new state of the network, it selects and executes another evidence-gathering action, and so on (see Fig. 4.1).

Figure 4.1: MUM Inference about here.

4.4.2 The ASK and MU systems

After building the MUM system, we abstracted its essential components and built an architecture called MU [Cohen *et al.*, 1987b]. MU has been used to

develop other systems like MUM that generate diagnostic workups via preferences. The central idea in MU is that decisions about actions are based on many *features* of a situation; for example, in medicine these include the cost, diagnosticity, risk, discomfort, and time required for a test; the number of supported disease hypotheses, the prior probabilities of these diseases, and whether any are serious; relationships between disease hypotheses such as causality or mutual exclusivity; relationships between actions and hypotheses, such as whether a test can discriminate two disease hypotheses; and contextual information such as whether the patient is feeble or robust.[6] MU allows knowledge engineers to rapidly define features and automatically updates them.

Features characterize the *states* of MU-based planners; for example, in one state we may have a feeble patient with suggestive evidence of a dangerous hypothesis and we may have available a completely diagnostic but risky test, and another moderately diagnostic treatment that protects against heart attack. Given this characterization of the choice between actions, MU requires preferences to tell it which to do. To be useful, preferences should be general. That is, preferences should *apply* at many points in diagnostic plans, and over many plans. A Ph.D. student in our laboratory has developed a technique for acquiring and then generalizing expert preferences [Gruber, 1987]. The method, called ASK for Aquiring Strategic Knowledge, uses a small set of preferences to generate diagnostic workups that experts then criticize. The criticisms are used to specialize the original preferences and to add new, specific ones that are not in the original set. These are generalized if later criticisms suggest their applicability is too narrow.

In sum, MU makes it easy to define and maintain the values of features and the ASK system makes it easy to define preferences based on features. Features and their values make up the dynamic state of the planner, and states and preferences determine which problem-solving actions will be taken.

The relationship between preferences and plans.

We rely on preferences to guide MUM's planning without any projection. Plans and preferences serve the same purpose, namely, to tell the problem solver what to do next. But plans require projection while preferences do not. We claim that, in MUM at least, plans and preferences are related in such a way that sequences of actions based on preferences will appear

[6] MUM did not incorporate all these features.

to be good plans. The simplest such relationship between preferences and plans is *preferences are local applications of the evaluation criteria for plans* (see Fig. 4.2). For example, we noted earlier that if we prefer inexpensive plans, then we can generate plans without projection by locally selecting the cheapest applicable action (Sec. 4.2.7). This will not always find the cheapest plan, but as a heuristic for guiding diagnostic workups, especially in combination with other preferences, it generates acceptable workups without the combinatorics of planning.

Figure 4.2: Plans vs. Preferences about here.

The relationship between plans and preferences is illustrated by the *workup graph* for chest pain in Figure 4.3. This is an explicit contingency plan for the diagnosis of angina, generated by an expert internist, and full of implicit preferences. Obviously, one *can* generate hundreds of other workup graphs for chest pain by taking actions in different orders; for example, one might generate a plan that puts angiogram before therapy. But this and other syntactically possible workups violate expert preferences.[7] By following these preferences, one can generate a sequence of actions reactively, one at a time, that look as if they were planned in advance (as discussed in Sec. 4.2.6) and concur with Figure 4.3.

Figure 4.3: Workup diagram about here.

[7]For example, doing an angiogram before prescribing therapy violates at least four preferences: First, therapy provides evidence about angina that is more *efficient*, that is, a bit less diagnostic than an angiogram but much, much less expensive in terms of dollars, pain, and risk; second, therapy has few *side-effects*; third, it provides *evidence about the relevance of later tests*, specifically evidence about whether the angiogram is necessary; fourth, therapy *extends the time* before the physician must take his or her "final action," which is surgery. Each of these is an argument, or preference, for therapy over an angiogram at a particular point in a workup.

Why we need both preferences and plans

Although MUM generates diagnostic plans from preferences alone, we believe that expert physicians plan, that is, project the outcomes of actions. They don't plan complete diagnostic workups, but they do some limited lookahead.[8] Here are two examples of lookahead in MUM's domain:

> *The Cascade Effect.* Some tests lead inexorably to others that you may want to avoid, and so should be avoided themselves. For example, you want to avoid a stress test if possible because unless the patient is absolutely cleared, you're obliged to go on to the next step, which is an angiogram. Now almost everyone has some degree of coronary artery blockage, and nobody knows how much is too much. So you start with a stress test that isn't conclusive, and then you find some blockage, and then you're forced to do surgery, even though the patient may not have coronary artery disease.
>
> *Dependent tests.* If test 1 is diagnostic but costs a lot, and test 2 costs less but provides lower-quality evidence, then you may plan to do test 2 first and only do test 1 if test 2 comes back positive.

In both these cases, the selection (or avoidance) of an action is based on projecting the outcomes of the actions.

These examples hint at two circumstances in which projection is desirable. In the cascade effect, projection detects *pitfalls*—situations where an attractive action leads later to an unattractive one. A more familiar example of a pitfall is shown in Figure 4.4: When presented with a queen for the taking, my preferences say go ahead. The pitfall is that the queen is a sacrifice and checkmate follows. My preferences *should* say "Prefer piece exchanges that end with me ahead," but this requires projection. MUM was able to plan without projection because its search space of plans contained almost no pitfalls; however, some diagnostic plans are more *efficient* than others. Efficiency is the other reason for projection. If the order of actions affects the efficiency of plans, as it does whenever actions are not independent (see Sec. 4.2.5), then projection will contribute to efficiency.

[8] The word lookahead suggests an analogy with game-tree search, in which preferences have the same role as static evaluation functions and projection is lookahead to some depth horizon. The analogy suggests reactive planners are at one end of a continuum (they evaluate actions at a horizon of one or zero) and that planners can be more or less reactive depending on where they draw their depth horizon.

In the following section, we describe a planning problem in which both pitfalls and efficiency are concerns. This problem involves real-time planning in highly uncertain environments. We are using it to study tradeoffs between projection and reaction.

Figure 4.4: A decision in Chess diagram about here.

4.4.3 PLASTYC: Planning in Real-Time, Dynamic Environments

We have built a large simulation of forest fires and the equipment commonly used to put them out. We are building a planner called PLASTYC that operates in this dynamic, real-time world. The planner's goal is to manage the fire—limit the loss of human life, limit the damage to forest and buildings, and limit the monetary costs of achieving these goals. This task closely approximates the problems faced by a forest fire manager, but more importantly, it is representative of a class of problems that we believe require both traditional planning and the kind of reactive planning we used in MUM. In this section we describe the characteristics of this class of problems and sketch the planner we are building. This work is in progress, so the conclusions of this section are tentative.

Our simulation consists of a large geographical area ("Explorer National Park") in which there is a considerable variety of topography and ground cover, as well as roads, lakes, and streams. These features affect how forest fires burn. Equally important features are wind speed and direction, both of which can change unpredictably, and the moisture content of the ground cover, which varies in time and geographically. To fight the fire, the simulation provides bulldozers, crews, transport vehicles, planes and helicopters. These cut fire line, move firefighters, spray water, or dump retardent.

These fire-fighting agents can be directed either by an automatic planner or else by a human player of what is essentially a complex, real-time video game. We have already gained considerable insight into (and respect for) the dynamics of this mini-world by playing against the simulation—often losing many lives and considerable real estate to a seemingly slow and containable fire. It is difficult for a planner, human or AI program, to do very well at

the game (i.e., put out the fire with reasonable costs, no loss of life, etc.) because:

- The player's knowledge of the fire is limited to what the agents in the field can "see." Crews and bulldozers can see only short distances; aircraft can see further. The planner rarely, if ever, has complete knowledge of the extent or location of the fire (see Fig. 4.5).
- The behavior of the fire cannot be accurately predicted because some factors that affect it (e.g., terrain, ground cover and the moisture content of the ground cover) are known only approximately. Moreover, wind speed and direction can change unpredictably.
- The behavior of the fire-fighting agents cannot be accurately predicted. In particular, the time required to move to a location or perform some task depends on terrain and ground cover. Fire-fighting agents also have limited autonomy to run away from a fire, so the central planner cannot always be sure of their location.
- The simulation is real-time with respect to the fire. While fire-fighting agents move, cut line and drop retardent, the fire keeps burning. Most important, any time the planner devotes to deliberation is claimed as real estate by the fire.

Figure 4.5: Fire maps figure about here.

The environment with which the planner interacts is independent, dynamic and probabilistic. It is independent because the planner is not the only agent of change, dynamic because changes take place over time, and probabilistic because the magnitude and temporal extent of changes to the environment are unpredictable.

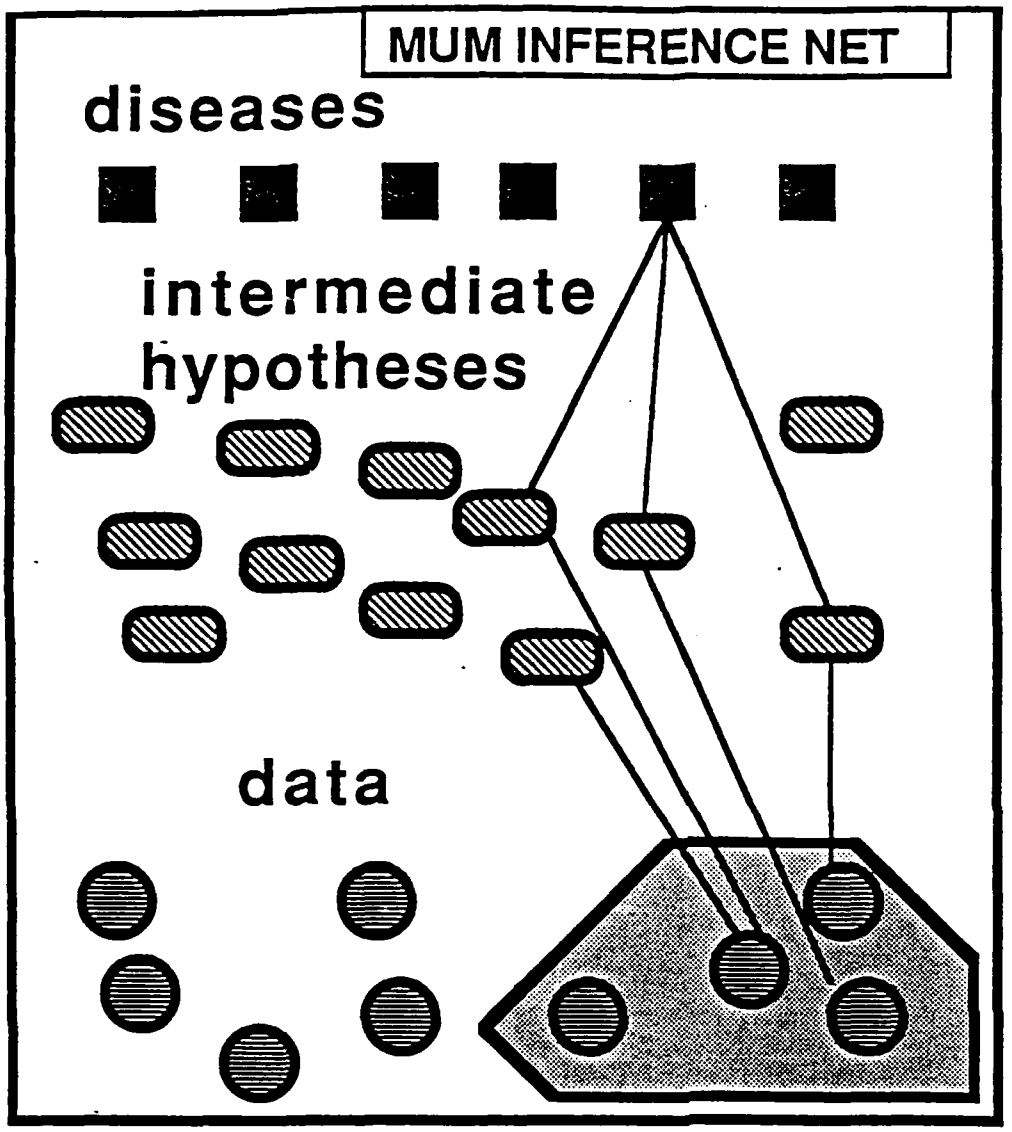
To plan in this environment an agent needs to know when to project and when to plan reactively. Projection is desirable to avoid pitfalls and to increase the efficiency of plans. But projection itself takes time, and uncertainty precludes avoiding *all* pitfalls, and efficient plans are useless if no time remains to execute them; so there will be situations when the timeliness of reaction will make the difference between success and failure.

We have described the desired behavior of PLASTYC, but not how the program will get the knowledge it needs to behave that way. We have become adept at putting out fires in the forest fire simulation, and we will encode this knowledge in PLASTYC. But this is slow. PLASTYC itself should acquire and refine these skills through practice. The extent to which this can be accomplished remains to be seen; the project is in its early stages.

4.5 Conclusion

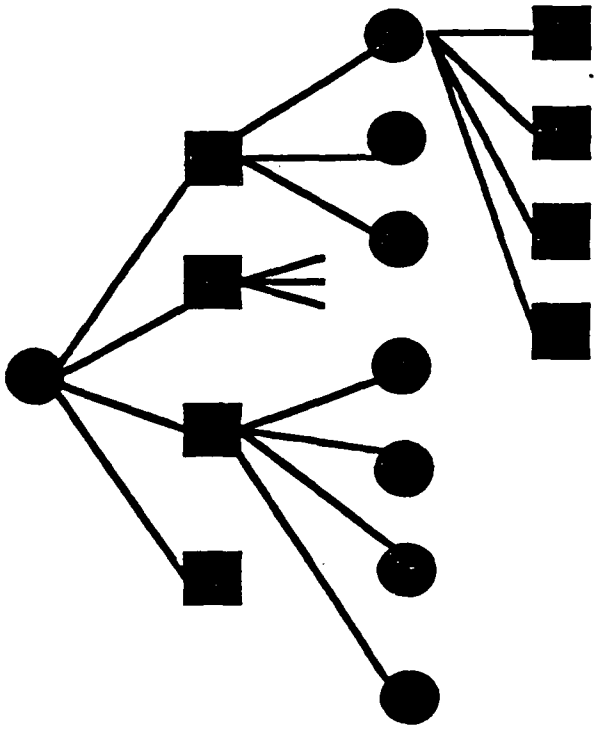
We argued in the Introduction that we should study reasoning under uncertainty in the context of autonomous action. In conclusion, we offer some methodological observations. In MUM, we wanted to study problem-solving strategies in medical diagnosis, but first we had to build an expert system, and then we had to acquire test cases from an expert. But we wanted to generate thousands of problems for our planner and to evaluate the planner on objective criteria. Neither was possible in internal medicine. We needed to challenge our planners with something like a game, but one that is played in a complex environment, in real time, under significant uncertainty. PLASTYC is designed to play autonomously against such a game—a simulation of forest fires. Not only does this simulated world provide an objective and efficient way to evaluate PLASTYC's abilities, but it will present thousands of individual problems from which PLASTYC will begin to learn.

Our experience with PLASTYC suggests a general method for addressing problems in approximate reasoning. If you can build a simulator that presents agents difficult problems in uncertain environments, and you can build agents that solve these problems autonomously, then you will have demonstrated unambiguous progress towards theories of action under uncertainty.

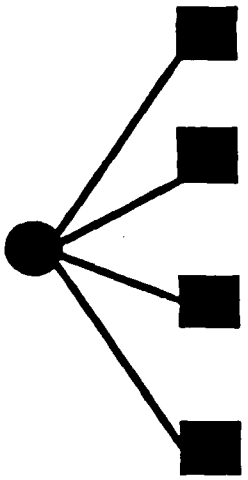


MUM: Determine focus of attention then select an evidence-gathering action

Figure 4.1.



subjective expected utility judgment based on cost, time, risk to patient diagnosticity, etc.



preference judgment based on cost, time, risk to patient, diagnosticity, etc.

Figure 4.2.

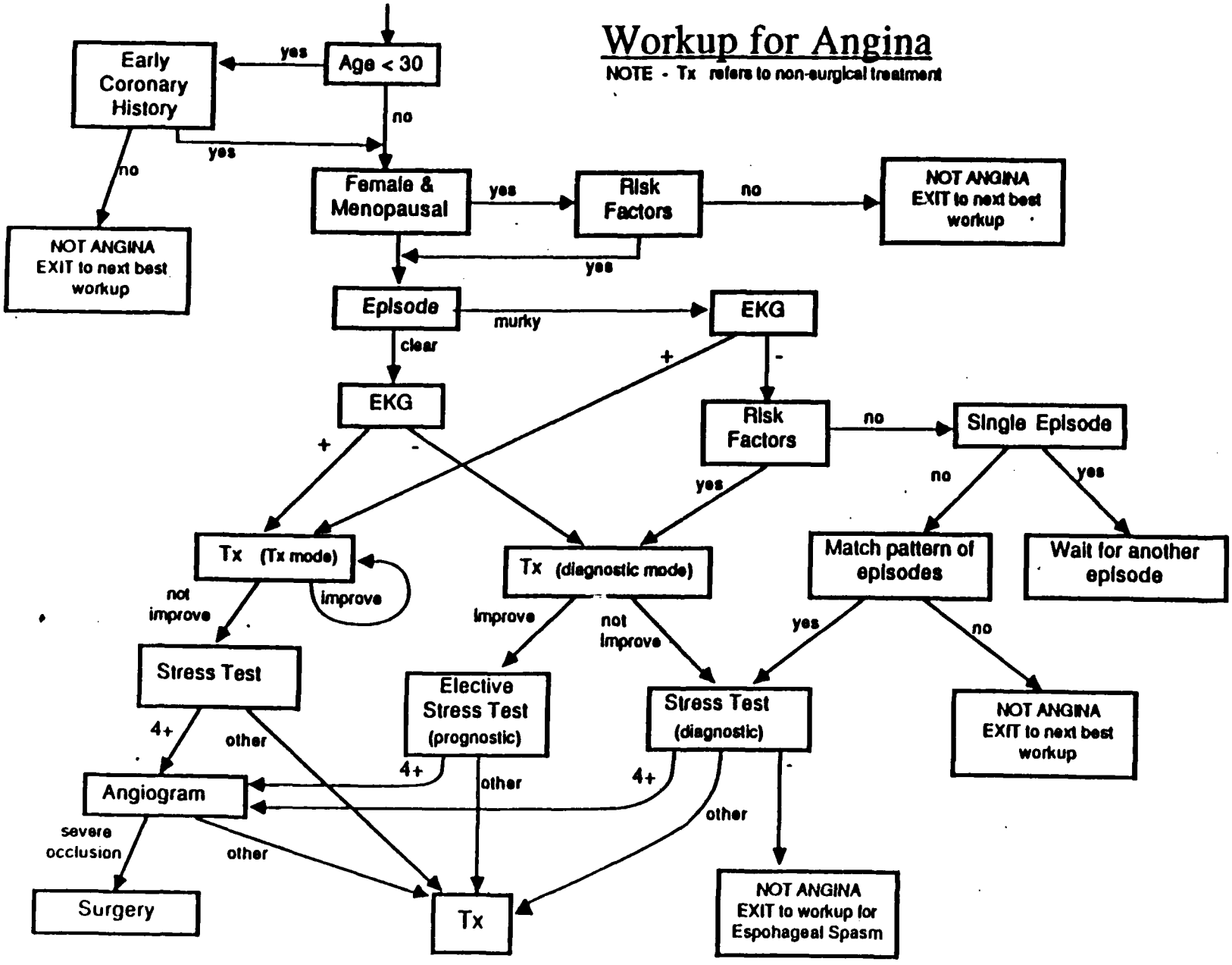

Workup for Angina
NOTE - Tx refers to non-surgical treatment
Age < 30
yes
no
Early Coronary History
yes
no
NOT ANGINA EXIT to next best workup
Female & Menopausal
yes
Risk Factors
no
NOT ANGINA EXIT to next best workup
yes
Episode
murky
EKG
clear
+
-
EKG
Risk Factors
no
Single Episode
+
-
yes
no
yes
Tx (Tx mode)
improve
not improve
Tx (diagnostic mode)
Match pattern of episodes
Wait for another episode
Improve
not Improve
yes
no
Stress Test
Elective Stress Test (prognostic)
Stress Test (diagnostic)
NOT ANGINA EXIT to next best workup
4+
other
4+
other
4+
other
Angiogram
severe occlusion
other
Surgery
Tx
NOT ANGINA EXIT to workup for Espohageal Spasm

Figure 4.3.

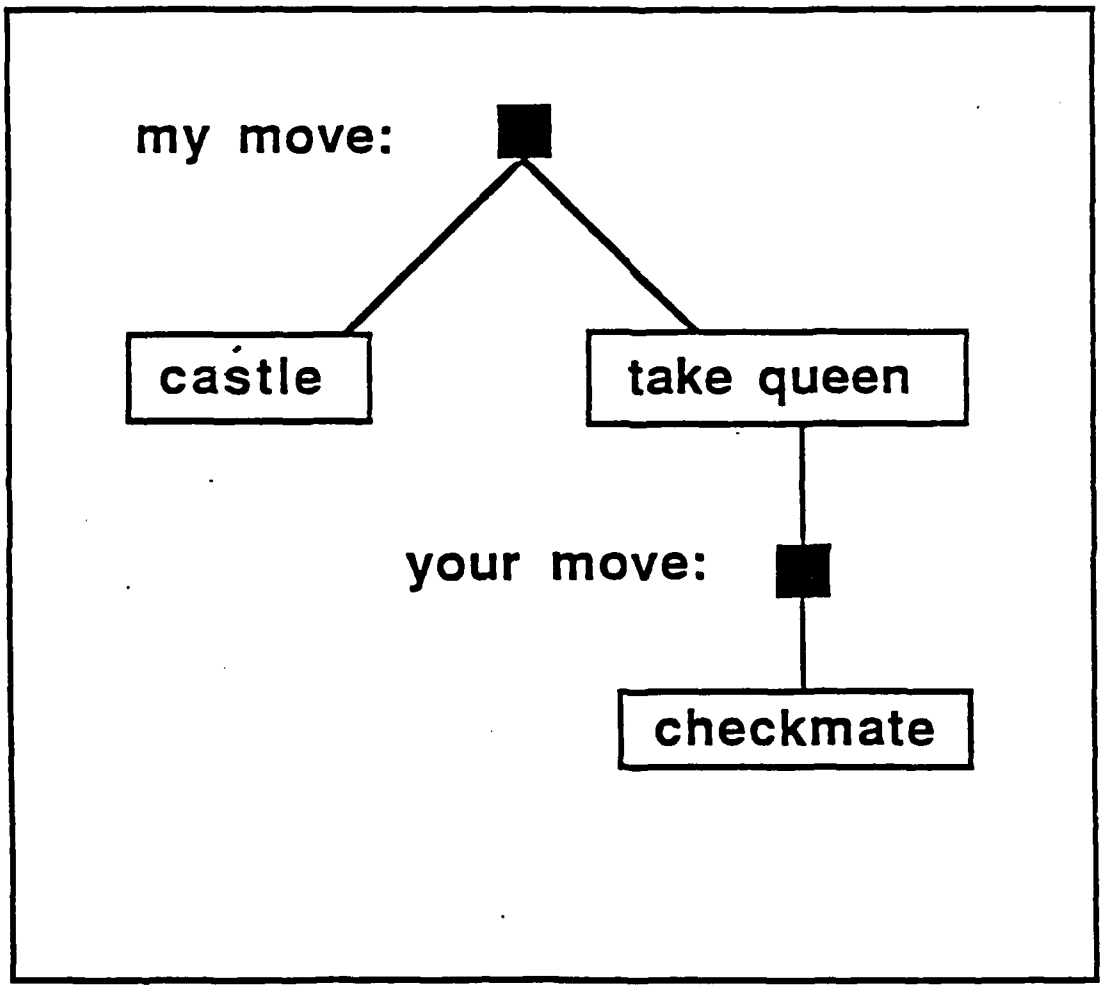


Figure 4.4.

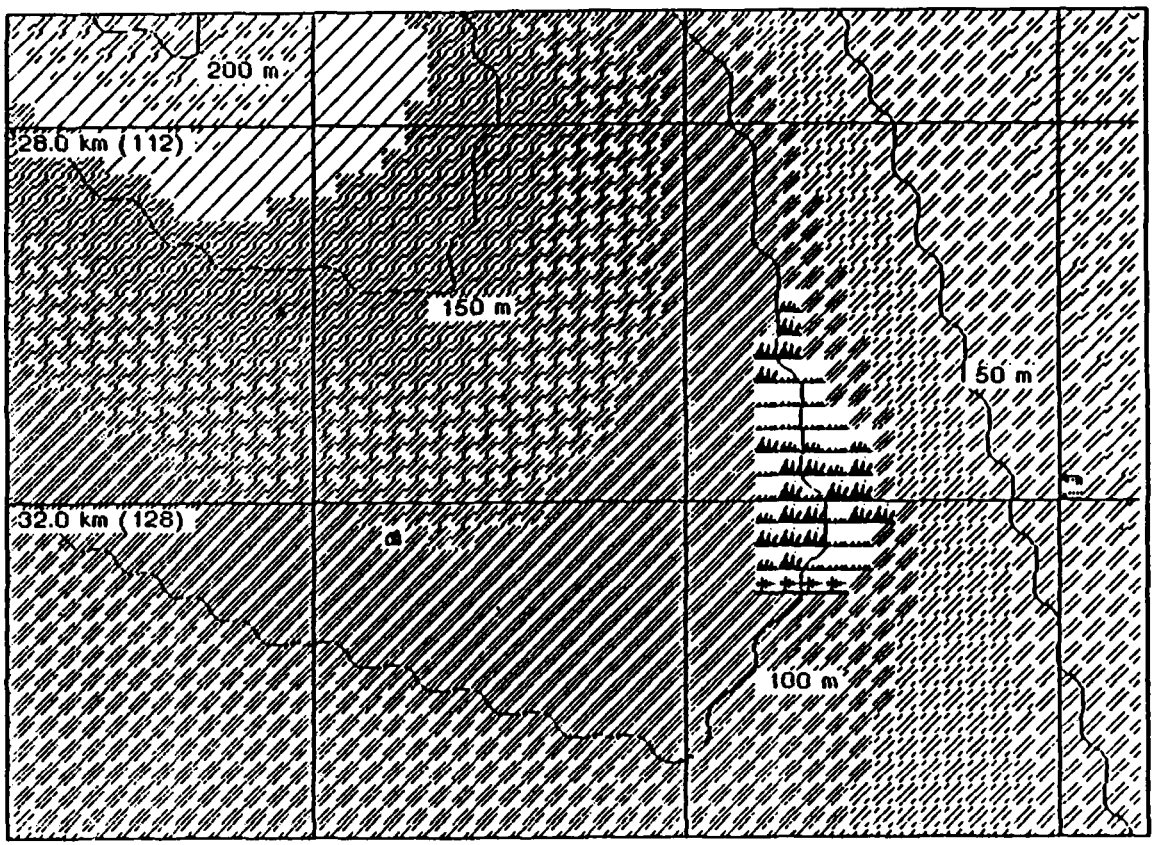


Figure 4.5A The state of the fire as it can be seen by the planner.

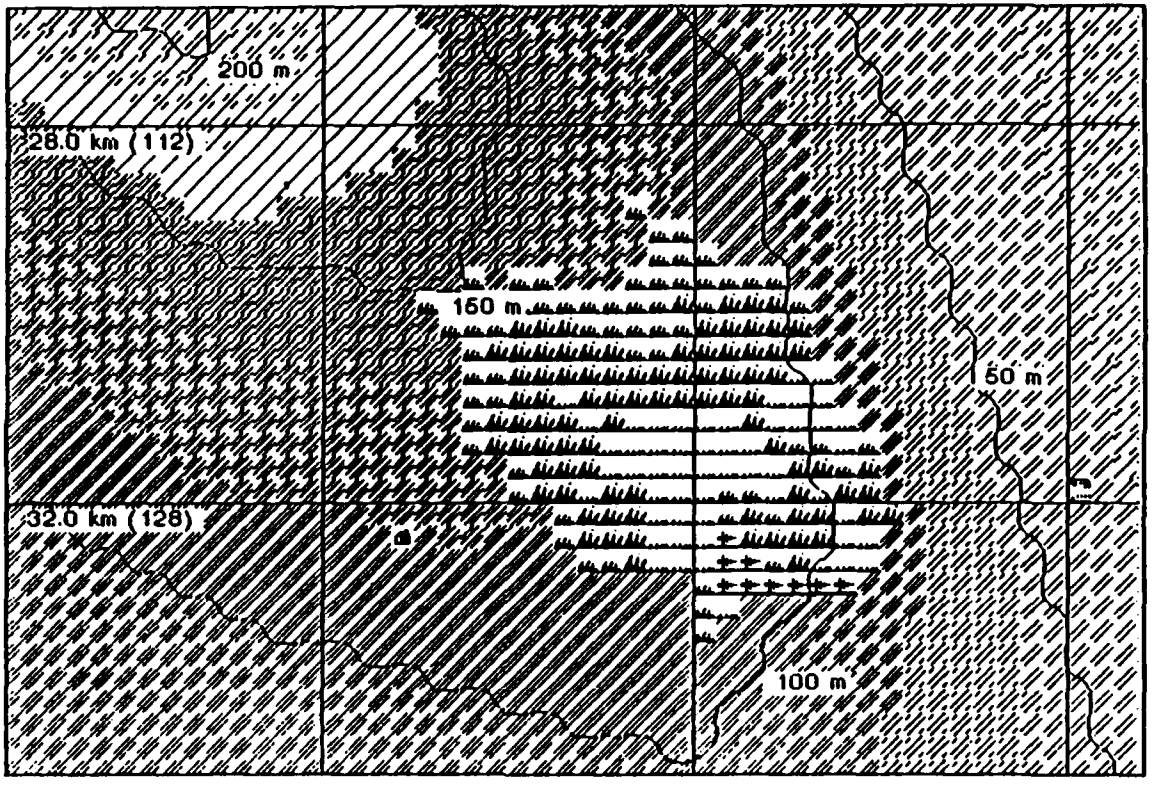


FIGURE 4.5B. The state of the fire as it really is.

Chapter 5

Trial by Fire: Understanding the Design Requirements for Agents in Complex Environments

5.1. The Phoenix Research Agenda

The Phoenix project is directed by three complementary goals. First, there are immediate technical aims: a real-time, adaptive planner for controlling simulated forest fires, approximate scheduling algorithms for coordinating multiple planning activities, knowledge representations for plans and for measuring progress toward goals, and distributed planning algorithms. Secondly, there are motivating issues, of which the foremost is to understand how complex environments constrain the design of intelligent agents. We seek general rules that justify and explain why an agent should be designed one way rather than another. The terms in these rules describe characteristics of environments, tasks and behaviors, and the architectures of agents. Lastly, because AI is still inventing itself, Phoenix is a commentary on the aims and methods of the field. Our position is that most AI systems have been built for trivial environments that offer no constraints on their design, and thus no opportunities to learn how environments constrain and inform system design [Cohen, 1989a]. To afford ourselves this opportunity, we began the Phoenix project by designing a real-time, spatially-distributed, multi-agent, dynamic, ongoing, unpredictable environment.

In the following pages we will describe Phoenix from the perspective of our technical aims and our motives. Section 5.2 describes the Phoenix task---controlling simulated forest fires---and explains why we use a simulated environment instead of a real, physical one. Section 5.3 discusses the characteristics of the forest fire environment and the constraints they place on the design of agents. The two lowest layers of Phoenix, described in Section 5.4, implement the simulated environment and maintain the illusion that the forest fire and agents are acting simultaneously. Above these are two other layers: a specific agent design (Sec. 5.5), and our organization of multiple fire-fighting agents (Sec. 5.6). These sections describe how Phoenix agents plan in real time, but do not provide minute detail . Section 5.7 is an example of Phoenix agents controlling a forest fire. Section 5.8 describes the current status of the project and our immediate goals.

5.2. The Problem

The Phoenix task is to control simulated forest fires by deploying simulated bulldozers, crews, airplanes, and other objects. We will discuss how the simulation works in Section 5.4, concentrating here on how it appears to the viewer and the problems it poses planners.

The Phoenix environment simulates fires in Yellowstone National Park, for which we have constructed a representation from Defense Mapping Agency data. Figure 5.1 shows a view of an area of the park; the grey region at the bottom of the screen is the northern tip of Yellowstone Lake. The thick grey line that ends in the lake is the Yellowstone River. The Grand Loop Road follows the river to the lake, where it splits. The large "B" in the bottom left corner marks the location of the fireboss, the agent that directs all others. Two bulldozers are shown building fireline around a fire in this figure.[1]

[1]Much available information is not displayed in the monochrome interface to Phoenix. The color interface displays ground cover and elevation contours, giving the user a better picture of the terrain over which the fire is spreading.

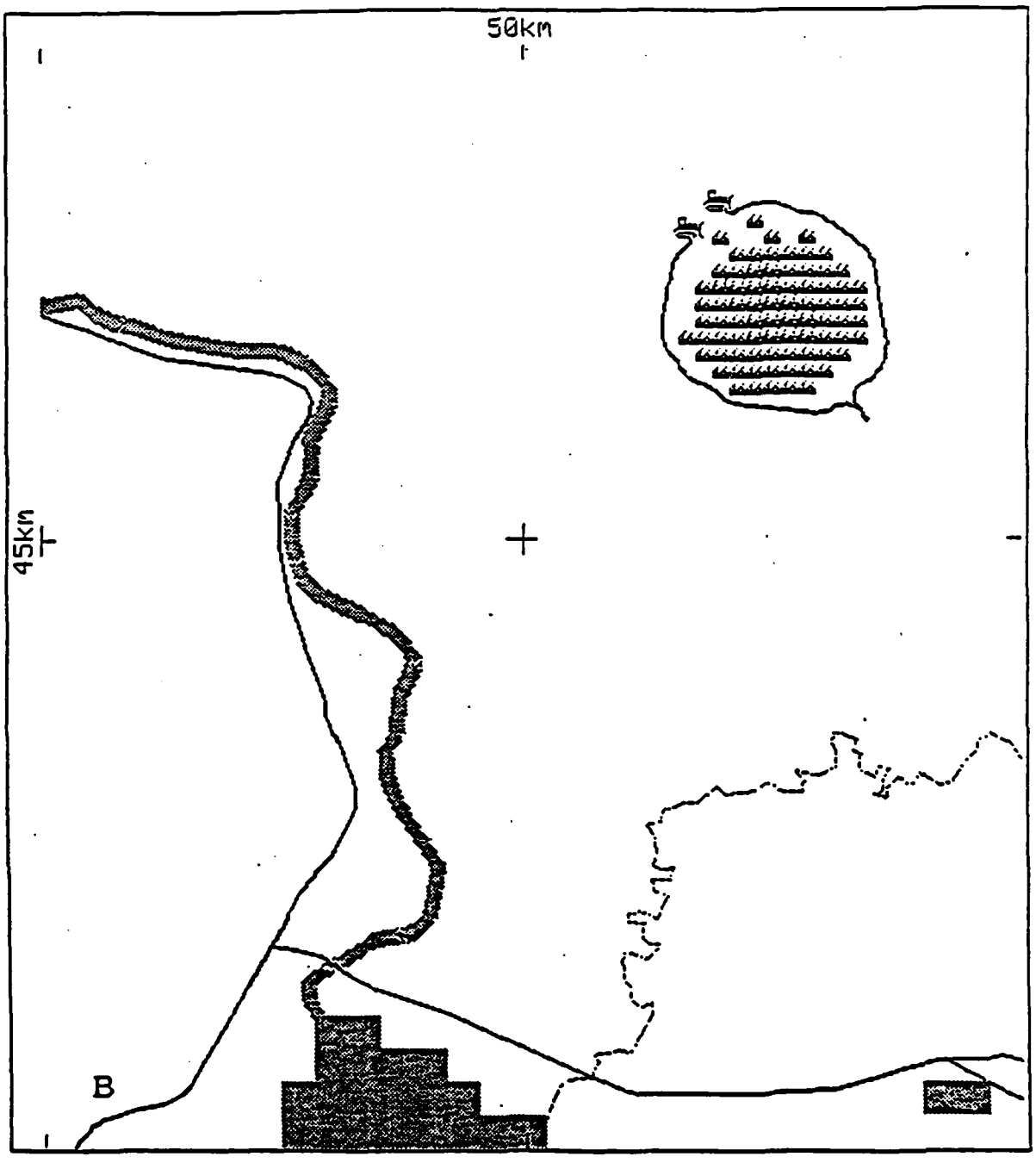


Figure 5.1: A portion of Yellowstone National Park as viewed in the Phoenix simulator. *This is a small fraction (about 7 km. by 8 km.) of the entire 75 km. square map. The northern tip of Yellowstone Lake appears at the bottom (gray shading represents water). The Yellowstone River empties into the lake here, as does a smaller stream called Pelican Creek (meandering line in lower right). Grand Loop Road runs along the lake and river from south to north. East Entrance Road cuts across above the lake from west to east. The large* B *marks the fireboss and bulldozer base. In this frame two bulldozers have almost surrounded a fire with fireline. The fire is burning at different intensities; the inner part has been burning longer and is hotter (note the darker icon). The kilometer markings in the margins show distances east and south from the northwest corner of the park.*

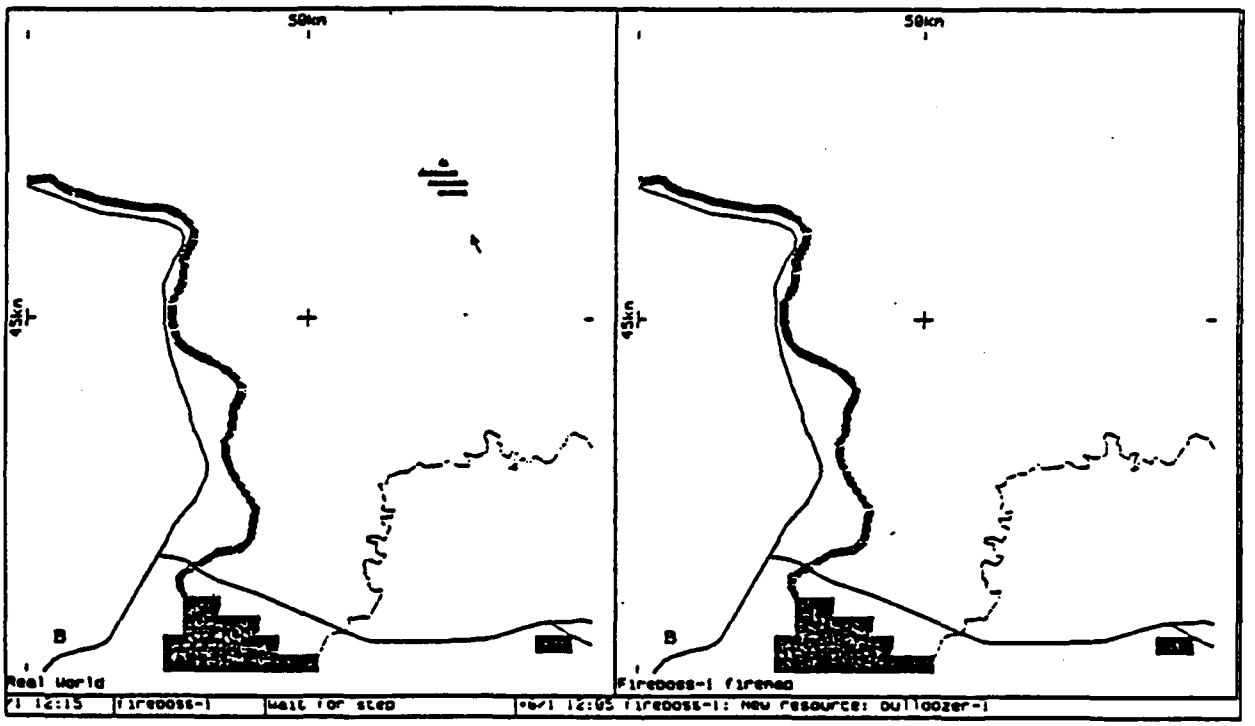


Figure 5.2: Fire at 12:15 pm

The left pane displays the real world; the right pane displays the current state of the world as the fireboss "sees" it. A fire has started and is displayed in the upper right of the real world pane. The fireboss, who finds out about fires from watchtower reports, doesn't know about this fire yet (see right pane).
The status bar below the two panes shows information about the running simulation. The date and time are in the left box (partly obscured). All simulations start at 12:00 noon on August 1. The right box displays timestamped information messages from various tasks.

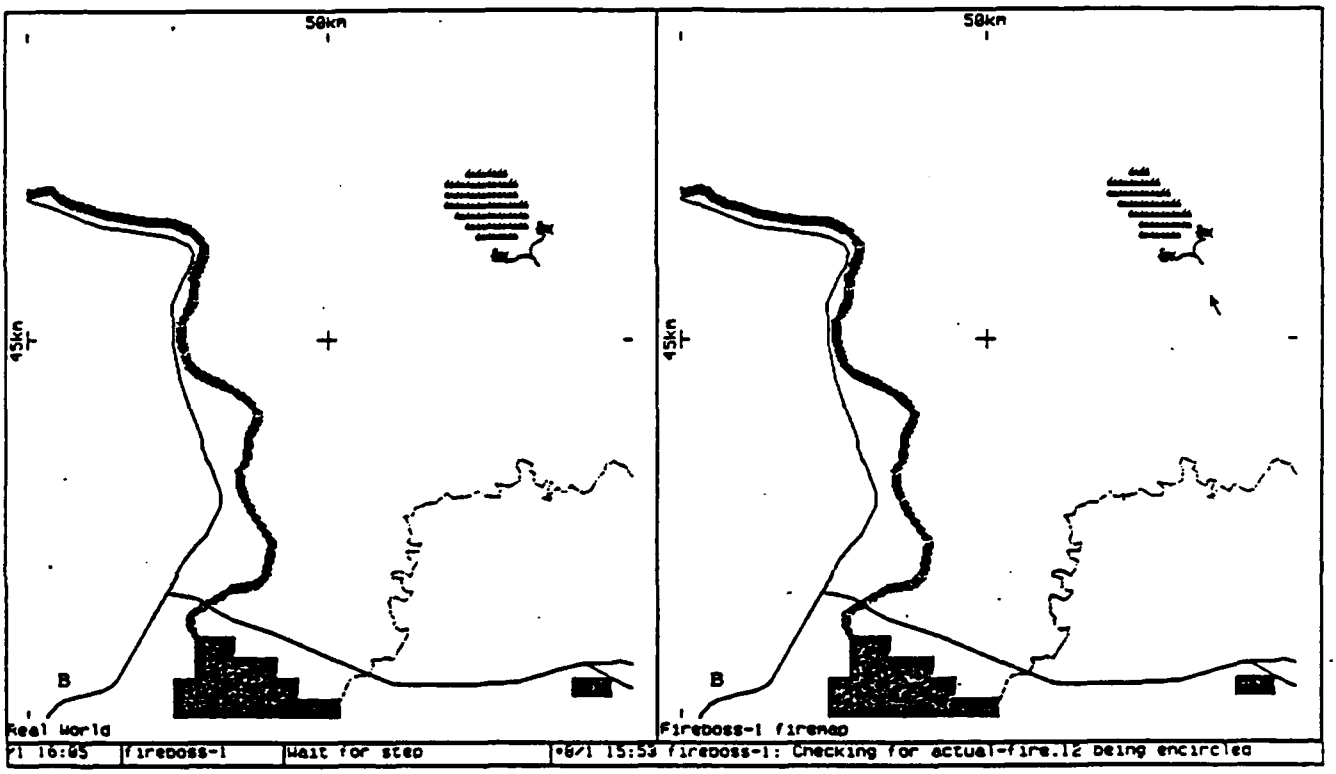


Figure 5.3: Fire at 4:05 pm

After four hours, two bulldozers have reached the fire and are beginning to build fireline around it. The two bulldozer plan was chosen by the fireboss based on environmental factors such as the size of the fire and the wind characteristics. Note that the fireboss's view of the situation is still slightly outdated. It sees fewer of the burning cells and isn't aware of all fireline that has been dug. It learns about these events from status reports sent by agents.

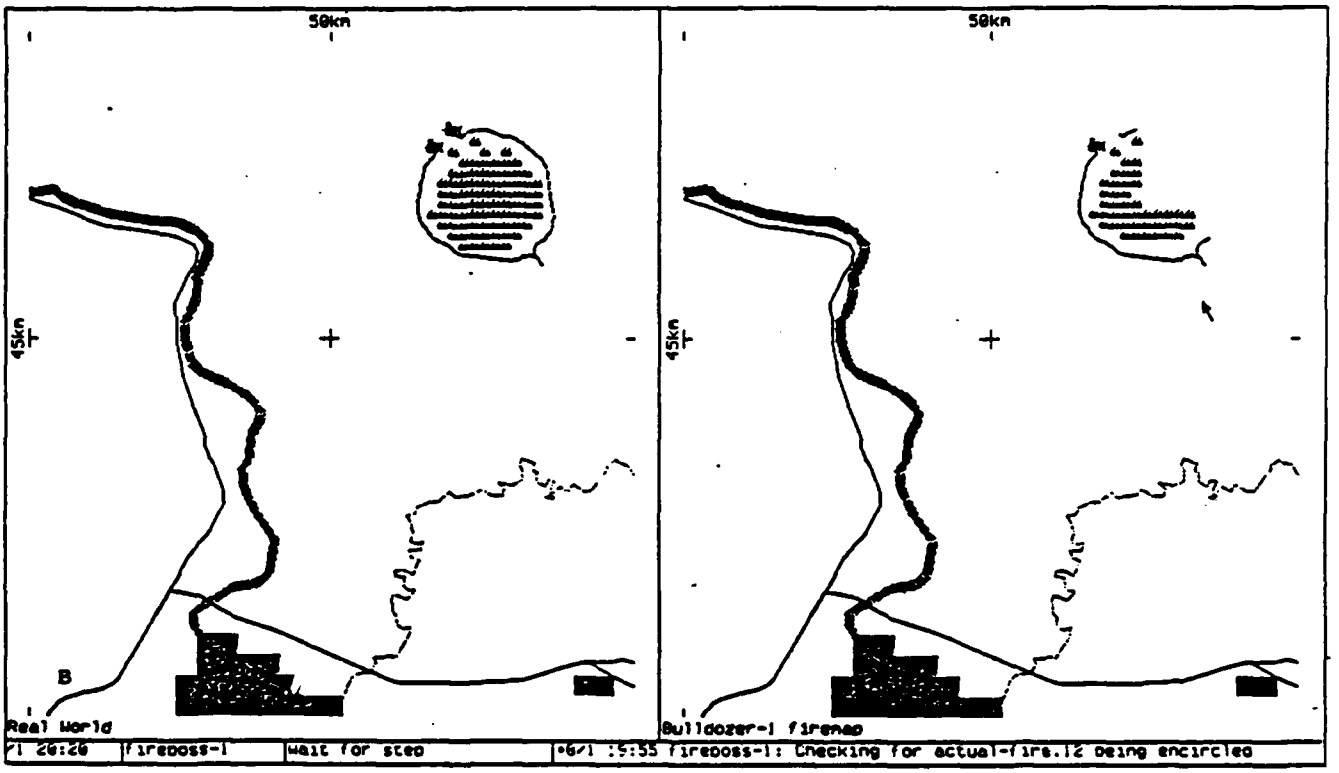


Figure 5.4: Fire at 8:20 pm

After 8 hours the fire is nearly encircled. The bulldozers are close to meeting at the fire front. The left pane again displays the real world; the right pane displays Bulldozer-1's view. It knows about the part of the fire it passed while digging line, as well as the fireline dug within its field of view (some of which was dug by the other bulldozer).

Fires spread in irregular shapes, at variable rates, determined by ground cover, elevation, moisture content, wind speed and direction, and natural boundaries. For example, fires spread more quickly in brush than in mature forest, are pushed in the direction of the wind and uphill, burn dry fuel more readily, and so on. These conditions also determine the probability that the fire will jump fireline and natural boundaries. But for two exceptional conditions (convective and crown fires), Phoenix is an accurate simulator of forest fires. Fire-fighting objects are also simulated accurately; for example, bulldozers move at a maximum speed of 40 kph in transit, 5 kph travelling cross-country, and 0.5 kph when cutting fireline. To give a sense of scale, the fire in Figure 5.1 is about 1.5 kilometers in diameter and has burned for about eight simulated hours. The fire's history, which can be read in Figures 5.2,5. 3, and 5.4 is as follows: At noon in simulation time (Fig. 5.2) a fire was ignited, and later detected by a watchtower (not visible in the figures). A little later, two bulldozers started a journey from the firestation, marked by a "B" in the southwest corner, to the rear of the fire. Because the wind was from the southeast, the rear was southeast of the fire. At 3 p.m. (Fig. 5.3) the bulldozers arrived at the rear and started cutting fireline. Figure 5.4 was generated at 8 p.m., simulation time. The fire was contained a little later. This entire simulation took about 1 minute on a TI Explorer.

Fires are fought by removing one or more of the things that keep them burning: fuel, heat, and air. Cutting fireline removes fuel. Dropping water and flame retardant removes heat and air, respectively. In major forest fires, controlled backfires are set to burn areas in the path of wildfires and thus deny them fuel. Huge "project" fires, like those in Yellowstone last summer, are managed by many geographically dispersed firebosses and hundreds of firefighters.

The current Phoenix planner is a bit more modest. One fireboss directs a few bulldozers to cut line near the fire boundary. We currently lack but are implementing "indirect" attacks, which exploit natural boundaries as firebreaks (such as the river in Fig. 5.1) and "parallel" attacks, which involve backfires. The Phoenix planner does use common fire-fighting plans, such as the *two bulldozer surround* illustrated in Figures 5.2, 5.3, and 5.4. In this plan, two bulldozers begin at the rear of the fire and work their way around to the front, pinching it off.

The Phoenix fireboss directs bulldozers but does not control them completely. In fact, the fireboss gives fairly crude directions, such as "go to location x,y," and individual agents decide how to interpret and implement them. Thus, bulldozers and other agents are semi-autonomous. Other organizational structures are enabled by increasing or decreasing the degree of autonomy; for example, an earlier fire planner, designed by David Day, had a single fireboss that controlled every action of all its agents. At the other extreme, we are working with Victor Lesser on a completely distributed version of Phoenix, in which agents negotiate plans in the absence of a single fireboss. We can experiment with different organizational structures because all agents have exactly the same architecture, and so each can assume an autonomous, semi-autonomous, or completely subservient role.

Although Phoenix agents and their environment are all parts of a large software system, we have designed them to give the impression of independent agents "playing against"

simulated forest fires, much as we would play a video game. In fact, early in the project, we built an interface to allow us, instead of an automated planner, to direct fire fighting agents. It required us to control several agents simultaneously, and demanded considerable foresight and planning. We found it impossible to control more than a couple of bulldozers in real time in the vicinity of the fire, so we gave bulldozers simple reflexes, enabling them to scurry away from encroaching fire. Since then, the basic style of interaction between the Phoenix environment and the Phoenix planners has not changed: One or more planners, AI or human, direct semi-autonomous agents to move around a map, building line around continuously burning fires.

The decision to develop and test Phoenix agents in a simulated environment is, to some, profoundly wrong. One argument is that by building the environment and the interface to agents, we risk deferring or ignoring difficult problems. For example, if we build a simulated agent that has a completely accurate internal map of its simulated environment and, when it moves, its "wheels" don't slip, then all its planning and acting can be dead-reckoning. Of course we *can* create trivial environments and develop techniques that won't work in real environments, but why would we? The point of using simulators is to create more realistic and challenging worlds, not to avoid these challenges. In response to the criticism that simulators can never provide faithful models of the real, physical world, we argue that the fire environment is a real-time, spatially-distributed, ongoing, multi-actor, dynamic unpredictable world - irrespective of whether it is an accurate model of how forest fires spread. As it happens, the fire environment is an accurate model of forest fires, but this isn't necessary for the environment to challenge our current planning technology. Moreover, we want to leave open the possibility of working in simulated worlds that are unlike any physical world that we have encountered.

The advantages of simulated environments are that they can be instrumented and controlled, and provide variety; all essential characteristics for experimental research. Specifically, simulators offer these advantages:

Control. Simulators are highly parameterized, so we can experiment with many environments. For example, we can change the rate at which wind direction shifts, or speed up the rate at which fire burns, to test the robustness of real-time planning mechanisms. Most important, from the standpoint of our work on real-time planning, is the fact that we can manipulate the amount of time an agent is allowed to think, relative to the rate at which the environment changes, thus exerting (or decreasing) the time pressure on the agent (Sec. 5.4).

Repeatability. We can guarantee identical initial conditions from one "run" to the next; we can "play back" some histories of environmental conditions exactly, while selectively changing others.

Replication. Simulators are portable, and so enable replications and extensions of experiments at different laboratories. They enable direct comparisons of results, which would otherwise depend on uncertain parallels between the environments in which the results were collected.

Variety. Simulators allow us to create environments that don't occur naturally, or that aren't accessible or observable.

Interfaces. We can construct interfaces to the simulator that allow us to defer questions we'd have to address if our agents interacted with the physical world., such as the vision problem. We can also construct interfaces to show things that aren't easily observed in the physical world; for example, we can show the different views that agents have of the fire, their radius of view, their destinations, the paths they are trying to follow, and so on. The Phoenix environment graphics make it easy to see what agents are doing and why.

5.3. Environmental Constraints on Agent Design

From the preceding descriptions of the Phoenix environment and tasks, one can begin to see the challenges they present to Phoenix agents. Our challenge, as researchers, is to design these agents for the Phoenix environment. The relationships between agent design, desired agent behaviors, and environment characteristics are clarified by what we call the behavioral ecology triangle, shown in Figure 5.5.

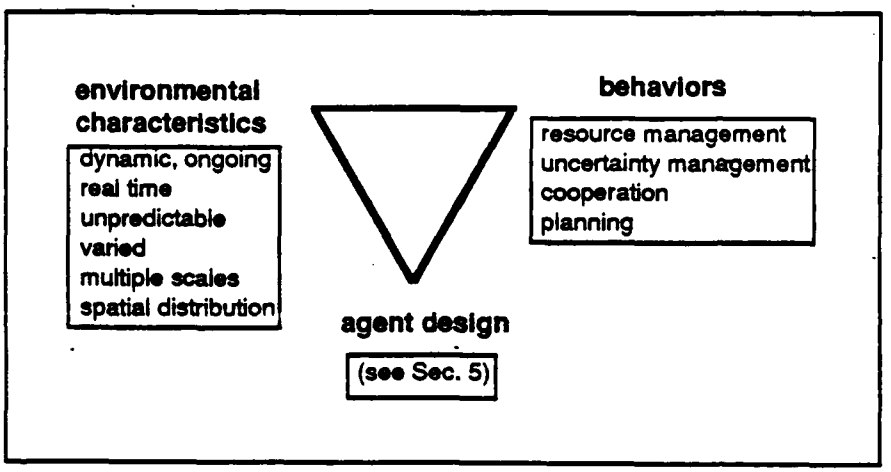


Figure 5.5: The behavioral ecology triangle.

The vertices of the triangle are the agent's design (i.e., internal structures and processes), its environment, and its behavior (i.e., the problems it solves and the ways it solves them). In this context, our tasks (and, indeed, the tasks of all AI research on intelligent agents) are:

Environment Analysis: What characteristics of an environment most significantly constrain agent design?

Design: What architecture will produce the desired behaviors under the expected range of environmental conditions?

Prediction: How will a particular agent behave in particular environmental conditions?

Explanation: Why does an agent behave as it does in particular environmental conditions?

Generalization: Over what range of environmental conditions can we expect particular behaviors from the agent? Over what range of problems? Over what range of designs?

To date, the Phoenix project has concentrated on environment analysis (see below), the design task (Secs. 5.5 and 5.6) and on building an environment in which the other tasks can be empirically pursued. Figure 5.5 implicitly captures many hypotheses and explanatory tasks. We can think of "anchoring" two corners and "solving for" a third; for example, we can anchor an environment and a set of behaviors and solve for an agent design. Or we can anchor a design and an environment and test predictions about behavior. Another more exploratory research strategy is to anchor just one corner, such as the environment, and look for tradeoffs in the other corners. For example, given the Phoenix environment, how is adaptability to changing time pressures affected by the design decision to search for plans in memory, rather than generate them from scratch?

Let us survey the characteristics of the Phoenix environment that constrain the design of Phoenix agents, and the behaviors the agents must display to succeed at their tasks. The fire environment is *dynamic* because everything changes: wind speed and direction, humidity, fuel type, the size and intensity of the fire, the availability and position of fire-fighting objects, the quantity and quality of information about the fire, and so on. The environment is *ongoing* in the sense that there isn't a single, well-defined problem to be solved, after which the system quits, but rather, there is a continuous flow of problems, most of which were unanticipated. The environment is *real-time* in the sense that the fire "sets the pace" to which the agent must adapt. The agent's actions, including thinking, take time, and during that time, the environment is changing. These characteristics require an agent to have some concept of relative or passing time. The agent must reason about the potential effects of its actions, and particularly about how much time those actions may require. Additionally, it must be able to perceive changes in its environment, either directly through its own senses or indirectly through communication with other agents.

The environment is *unpredictable* because fires may erupt at any time and any place, because weather conditions can change abruptly, and because agents may encounter unexpected terrain, or fire, or other agents as they carry out plans. An agent must respond to unexpected outcomes of its own actions (including the actions taking more or less time than expected) and to changes in the state of the world. This requires interleaving planning, execution and monitoring and suggests that detailed plans of long duration will be likely to fail before successful completion. The unpredictability of the environment requires agents to be flexible, particularly in the way they handle temporal resources. In fact, all resources, including time, fire fighting agents, money, fuel, and equipment, are limited and non-renewable. Because the environment is ongoing, decisions about resources have long-term effects that constrain later actions, and require agents to manage their resources intelligently, with a global perspective. For this reason, among others, Phoenix agents cannot be exclusively "reactive."

Whereas unpredictability is a characteristic of the Phoenix environment, *uncertainty* arises in agents. Uncertainty is partly due to the fire continuously moving, partly because changes in wind speed and direction are unpredictable, partly due to communication delays between agents, and partly because individual agents have very limited views of the

the world. For example, to the northeast of Bulldozer 1, in the right-hand pane of Figure 5.4, there is a small black patch of fireline. This is all Bulldozer 1 knows about the location and progress of the other bulldozer (whose actual location is shown in the left-hand pane of Fig. 5.4), and illustrates how far Bulldozer 1 can see. It follows that Bulldozer 1's firemap, as shown in the right-hand pane, must merge what it currently sees with what it recalls. As one would expect, the recollection is inaccurate; Bulldozer 1 thinks the fire at its southern point is a few hundred meters from the fireline, because that's where it was when Bulldozer 1 cut the fireline. In fact, the fire has spread all the way to the fireline, as shown in the left-hand pane. As a consequence of these types of uncertainty, agents must allot resources for information gathering. Agents must be able to integrate and disseminate local information, and, because of their own localized views, they must be able to communicate and coordinate with each other.

The fact that events happen at *different scales* in the Phoenix environment has profound consequences for agent design. Temporal scales range from seconds to days, spatial scales from meters to kilometers. Agents' planning activities also take place at disparate scales; for example, a bulldozer agent must react quickly enough to follow a road without straying due to momentary inattention, and must also plan several hours of fire-fighting activity, and must do both within the time constraints imposed by the environment.

Given the size and variation in the world map, the degree to which the environment can change, and the possible actions of agents, the environment can produce a large *variety* of states. Consequently, an agent must know how to act in many different situations. The ramifications for agent design depend on whether small differences in environmental conditions can produce large differences in the utilities of plans. For example, if every fire scenario is truly different in the sense that each requires a unique, scenario-specific plan, then it may be pointless to provide agents with memories of previous plans. In fact, we believe that although the fire environment presents a wide variety of states, these differences do not require radically different plans.

The Phoenix environment is *spatially distributed*, and individual agents have only limited, local knowledge of the environment. Moreover, most fires are too big for a single agent to control; their perimeters grow much faster than a single agent can cut fireline. These constraints dictate multi-agent, distributed solutions to planning problems. They also expand the scope of our research from a study of *agent* design to a study of *organizational* design. We have drawn a line, temporarily, and excluded the latter.

In sum, to perform their designated tasks, under the constraints of the Phoenix environment, Phoenix agents must engage in particular behaviors. In gross terms, these are resource management, uncertainty management, planning, and cooperative problem-solving; more specific behaviors have just been discussed. The question we address in Sections 5.5 and 5.6 is how do the characteristics of the Phoenix environment, in concert with the desired behaviors of Phoenix agents, constrain the design of the agents? Specifically, what architecture is capable of planning in real time, responding to events at different time scales, coordinating the efforts of several agents, collecting and integrating data about a changing environment, and so on.

5.4. The Phoenix Environment, Layers 1 and 2.

To facilitate experiments, Phoenix is built in four layers. The lowest is a task coordinator that maintains the illusion of simultaneity among many cognitive, perceptual, reflexive and environmental processes, on a serial machine. The next layer implements the Phoenix environment itself---the maps of Yellowstone National Park, and the simulations of fires. The third layer contains the definitions of the components of agents---our specific agent design. The fourth layer describes the organization of agents, their communication and authority relationships. Layers 3 and 4 are described in later sections.

The two lowest layers in Phoenix, called the *task coordinator layer* and *map layer* respectively, comprise the Phoenix discrete event simulator. We discuss the task coordinator first. It is responsible for the illusion of simultaneity among the following events and actions:

Fires: Multiple fires can burn simultaneously in Phoenix. Fires are essentially cellular automata that spread according to local environmental conditions, including wind speed and direction, fuel type, humidity, and terrain gradient.

Agents' physical actions: Agents move from one place to another, report what they perceive, and cut fireline.

Agents' "internal" actions: Internal actions include sensing, planning, and reflexive reactions to immediate environmental conditions.

These tasks are not *generated* at the task coordinator level of Phoenix, just scheduled on the cpu there. Fire tasks are generated at the map layer, and agent tasks are generated at the levels described in Sections 5.5 and 5.6.

Typically, the task coordinator manages the physical and internal actions of several agents (e.g., one fireboss, four bulldozers, and a couple of watchtowers), and one or more fires. The illusion of continuous, parallel activity on a serial machine is maintained by segregating each process and agent activity into a separate task and executing them in small, discrete time quanta, ensuring that no task ever gets too far ahead or behind the others. The default setting of the synchronization quantum is five minutes, so all tasks are kept synchronized to within five minutes of each other. The quantum can be increased, which improves the cpu utilization of tasks and makes the testbed run faster, but this increases the simulation-time disparity between tasks, magnifying coordination problems such as communication and knowing the exact state of the world at a particular time. Conversely, decreasing the quantum reduces how "out of synch" processes can be, but increases the running time of the simulation.

The task coordinator manages two types of time: *cpu time* and *simulation time*. CPU time refers to the length of time that processes run on a processor. Simulation time refers to the "time of day" in the simulated environment. Within the predefined time quantum, all simulated parallel processes begin or end at roughly the same simulation time. To exert real-time pressure on the Phoenix planner, every cpu second of "thinking" is followed by K simulation-time minutes of activity in the Phoenix environment. Currently K = 5, but this parameter can be modified to experiment with how the Phoenix planner copes with different degrees of time pressure.

The fire simulator resides at Phoenix's map layer; that is, the map layer generates tasks that, when executed by the task coordinator, produce dynamic forest fires. Phoenix's map, which represents Yellowstone National Park, is a composite of several two dimensional structures, and stores information for each coordinate about ground-cover, elevation, features (roads, rivers, houses, etc.), and fire-state. The fire itself is implemented as a cellular automaton in which each cell at the boundary decides whether to spread to its neighbors, depending on the local conditions just mentioned and global conditions such as wind speed and direction (currently, we do not model local variations in weather conditions). These conditions also determine the probability that the fire will jump fireline and natural boundaries.

The Phoenix discrete event simulation is generic. It can manage any simulations that involve maps and processes. For example, we could replace the forest fire environment with an oil-spill environment. We could replace our map of Yellowstone with oceanographic maps of, say, Prince William Sound. Fire processes have spatial extent, and spread according to wind speed, direction, fuel type, terrain, and so on. They could easily be replaced with oil-slick processes, which also have spatial extent, and spread according to other rules. Similarly, we could replace the definitions of bulldozers and airplanes with definitions of boats and booms.

5.5. Agent Design, Layer 3.

The third layer of Phoenix is our specific *agent design*, which is constrained by forest fire environment as described in Section 5.3. For example, because events happen at two dramatically different time scales, we designed an agent with two parallel and nearly-independent mechanisms for generating actions (Figure 5.6). One generates reflexive actions very quickly—on the order of a few seconds of simulated time—and the other generates plans that may take hours of simulated time to execute. This longer-term planning can be computationally intensive, because it incurs a heavy time penalty for switching contexts when interrupted. For this reason, the cognitive component is designed to do only one thing at a time (unlike sensors, effectors, or reflexes, where multiple activities execute in parallel). Both the cognitive and reflexive component have access to sensors, and both control effectors, as shown in Figure 5.6.

The agent interacts with its environment through its sensors and effectors, and action is mediated by both the reflexive and the cognitive components. Sensory information may be provided autonomously or may be requested, and sensors' sensitivity may be adjusted by the cognitive component. Effectors produce actions in the world such as information gathering, building fireline, and moving.

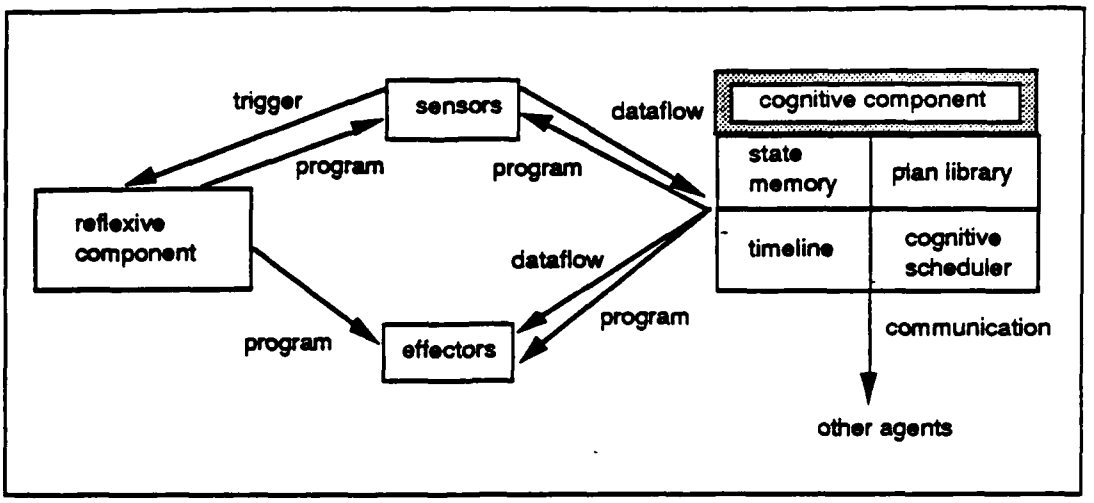


Figure 5.6: Phoenix agent design.

Reflexes are triggered by output of sensors. They change the programming of effectors to prevent catastrophes, or they fine tune the operation of effectors. For example, a bulldozer is stopped by a reflex if it is about to move into the fire, and reflexes handle the fine tuning necessary for the bulldozer to follow a road. Reflexes are allotted almost no cpu time, and have no memory of events, so they cannot produce coordinated sequences of actions. They are designed for rapid, unthinking action. Although some researchers have suggested that longer-term plans can *emerge* from compositions of reflexes [Agre and Chapman 1987, Brooks 1986], we do not believe that compositions of reflexes can handle temporally-extensive planning tasks such as resource management, or spatially-extensive tasks such as path planning with rendezvous points for several agents. Thus, we have adopted a design in which reflexes handle immediate tasks and a cognitive component handles everything else.

The cognitive component of an agent is responsible for generating and executing plans. Instead of generating plans de novo, as classical hierarchical planners did, the Phoenix cognitive component instantiates and executes stored skeletal plans. We believe this is a good design for the forest fire environment because, first, a relatively small number of skeletal plans is probably sufficient to cope with a wide range of fires; and, second, the store/recompute tradeoff suggests relying on stored plans, rather than computing them, in real-time situations. In addition to controlling sensors and effectors, the cognitive component handles communications with other agents (including integrating sensor reports), and it responds to flags set when reflexes execute. It also engages in a wide range of "internal" actions, including projection (e.g., where will the fire be in 20 minutes?), plan selection and scheduling, plan monitoring, error recovery, and replanning. Our implementations of some of these capabilities are quite rudimentary, and leave much room for improvement, as we discuss in Section 5.8.

In overview, this is how the cognitive component works: in response to a situation such as a new fire, an appropriate plan is retrieved from the *plan library* and placed on the *timeline* (Fig. 5.6). *State memory* stores information, such as weather, resource conditions, and sensory input, that helps the cognitive agent select appropriate plans and

instantiate the variables of the chosen plan for the current situation. For example, if the fire is small and nearby, and the weather is calm, then a one-bulldozer plan will be retrieved and instantiated with situation-specific information such as the wind speed and the current location of the fire. The actions in a plan are eventually selected for execution by the *cognitive scheduler*, described shortly. At any time during this process, sensory data may trigger reflexive actions; for example, if the cognitive component is executing a command to move to a destination, and a sensor reports fire ahead, then the reflexive component will send a command to reverse direction. This happens very fast relative to the cycle time of the cognitive component, so the reflexive component sets a flag to tell the cognitive component what it did. When the cognitive component notices the flag, it might modify its plan. The analogy here is to our own reflexes, which yank us away from hot surfaces long before our cognitive apparatus becomes aware of the problem.

With this overview in mind, let us consider the operation of the cognitive component in detail. We will focus on the operation of the *fireboss* agent, which plans the activities of other agents such as bulldozers and crews. Each of these, in turn, plans how to carry out the directives of the fireboss. Because bulldozers and crews have the same architecture as the fireboss (Fig. 5.6), they can reason in exactly the same way. In the following discussion, we first describe planning when things go according to plan, and then describe error handling, interruptions, and other unexpected events.

When a fire is reported, an *action* called "deal with fire" is retrieved from the plan library and used to create a *timeline entry*, in this case called "deal with fire 27", which is added to the timeline (see Figure 5.7). Actions are general representations of the cognitive activities the agent can perform, such as path planning or communication, and describe applicability conditions, resource constraints and uninstantiated variables. Creating a timeline entry instantiates an action: binding its variables and adding the temporal constraints that relate it to other actions the agent has chosen to execute. Although timeline entries represent actions, it is not quite accurate to say they are executed (although we will use this terminology where the accurate description is too awkward). In fact, when a timeline entry is created, it inherits a set of *execution methods* from the action it instantiates. Each of these methods will execute the desired action; they differ along dimensions such as the time they require and the quality of their outputs. For example, a single action "plan a path" points to several path-planning algorithms, some which run quickly and return adequate paths, and some that run longer but produce shorter paths. When a timeline entry is selected for execution, the execution method most appropriate to the current circumstances is chosen. By delaying the choice of methods, the cognitive scheduler can reason about its own use of time, and select execution methods that are suited to emerging time constraints.

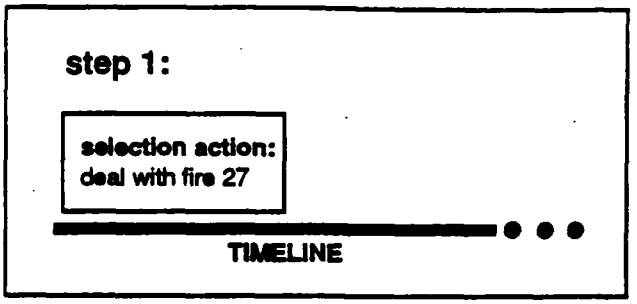


Figure 5.7: Contents of fireboss's timeline after being notified of a new fire: action to search for a plan to deal with the fire.

If there are entries on the timeline (e.g., "deal with fire 27") then the cognitive scheduler of the Phoenix cognitive component makes three decisions:

Which action to execute next

How much time is available for its execution

What execution method should implement the action

The cognitive scheduler always selects the "next" action on the timeline to execute, but often, several actions have this distinction and a choice must be made. Actions on the timeline may be unordered (and thus equally entitled to "go first") for several reasons: skeletal plans often leave actions unordered so that the cognitive scheduler has flexibility at execution time to select the best order. Or, frequently, the agent is executing several plans simultaneously. This happens, for example, when several fires are reported. The planner formulates plans for each, but doesn't specify temporal constraints among actions from different plans. In the current example, however, the only action on the timeline is "deal with fire 27," so the cognitive scheduler determines how much time is available to execute it and selects an execution method. In this case, it selects a method called *find and filter plan* (step 2, Fig. 5.8). Its effect, when executed, is to search the plan library for a plan to "deal with fire 27." First it finds all plans for dealing with fires of this type, then it filters the infeasible ones, then selects from the candidates to find the most appropriate one, and lastly, it adds a new action to the timeline called "2 BD surround." (This plan, illustrated in Figs. 5.2-5.4, involves sending two bulldozers to a rendezvous point, then to the fire, after which they cut fireline in opposite directions around the fire.)

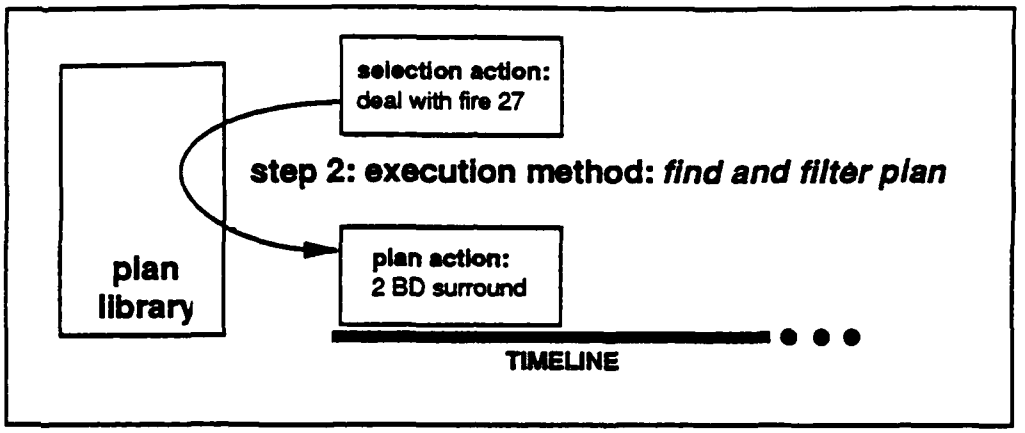


Figure 5.8: The fireboss executes timeline action, deal with fire 27, which searches the plan library, selects the 2 BD surround plan as appropriate for dealing with new fire, and places the new plan on the timeline.

Once again, the cognitive scheduler selects an action (the only one is "2 BD surround") assesses how much time is available, and selects an execution method. In this case, the method is *expand plan*. The result is to add a network of actions, partially ordered over time, to the timeline (step 3, Fig. 5.9). The network starts with a placeholder action, s, followed by two unordered actions that allocate bulldozers 1 and 2 respectively. The next action determines the rendezvous point for the bulldozers. Then two unordered actions bind the variables in the plan with the current wind direction and the previously-determined rendezvous point. Space precludes showing the rest of the plan in Figure 5.9.

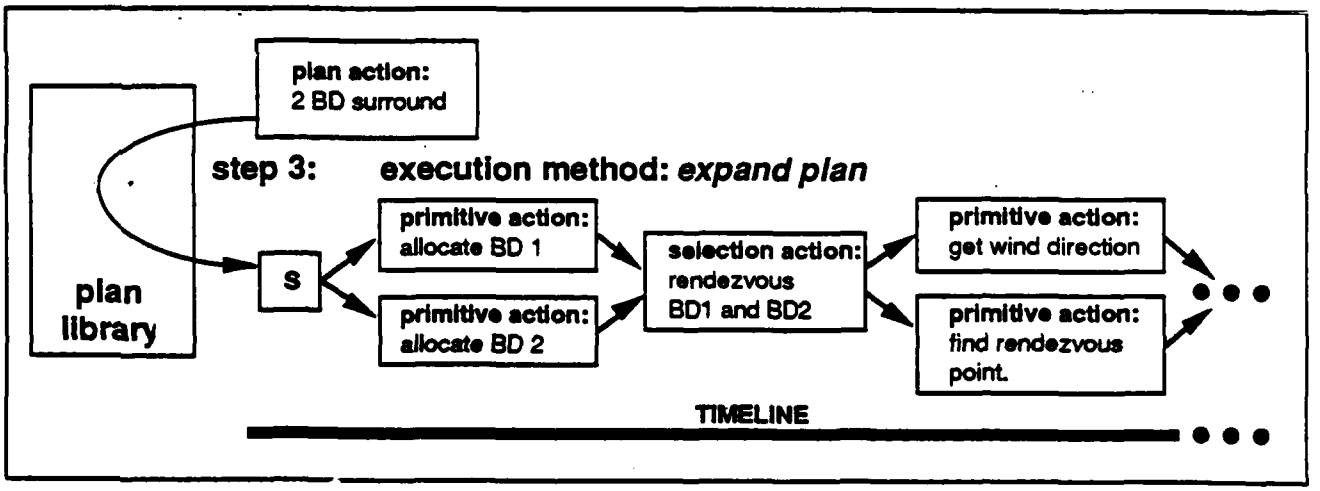


Figure 5.9: The fireboss executes timeline action, 2 BD surround, which expands into a network of plan steps.

The cognitive scheduler again looks at the timeline, and now must make a decision about which action to select. The "allocate bulldozer" actions are unordered, so one must be selected to go first. Then, as before, the cognitive scheduler assesses the available time

and selects an execution method. We will leave this example here, and discuss it further in Section 5.7.

Three kinds of actions can be differentiated by their effects on the timeline when they are executed: *selection* actions, like "deal with fire 27" result in a search of the plan library, after which a *plan action* such as "2 BD surround" is posted on the timeline. Plan actions are placeholders for plans; executing them results in plan expansions being posted on the timeline. Many of the actions in a plan are of the third type: *primitive* actions that result in a computation (e.g., calculating a route), or a command to a sensor or effector. But a plan can contain any of the three types of actions; for example, the expansion of "2 BD surround" contains a selection action. When executed, it will result in a search of the plan library for a plan to rendezvous the two bulldozers. Plans can also contain plan actions, which, when executed, add subplans to the network. This is our mechanism for representing hierarchical plans. Lastly, plans may contain just a single, primitive action, such as finding the rendezvous point for two bulldozers.

We have discussed how actions are scheduled and executed when everything goes according to plan, but in the Phoenix environment it rarely does. Phoenix agents have three abilities, all rudimentary, to handle unexpected events. Reflexes, operating on a very short time scale, can halt or modify potentially injurious actions, such as straying into the fire. By design, reflexes do very little processing, and return very little information. When a reflex halts a bulldozer, it simply posts a flag for the cognitive component; it does not interrupt the cognitive component to explain what it did. The cognitive component doesn't become aware of the change until it executes a regularly-scheduled status-checking action. In fact, by design, nothing ever interrupts a cognitive action. This is because the cost of saving state and switching context is prohibitive. Instead, the reflexive component of a Phoenix agent is expected to deal with situations as they arise. Most, like staying parallel to a moving fire, will never require the attention of the cognitive component anyway; but even when a serious problem comes up, the reflexive component is designed to keep the agent functioning until the cognitive component finishes its current task.

The second mechanism for handling unexpected situations is error recovery and replanning. Errors are unexpected events that preclude completion of an action or a plan. For example, bulldozers will travel to their designated destinations but fail to find a fire, path planning will sometimes fail to generate a path, selection actions will search the plan library but fail to find a plan that satisfies all constraints, and so on. Currently, over a dozen types of error can arise in Phoenix, although we don't have plans to deal with them all yet. The error handling mechanism is to post on the timeline a "deal with error" selection action, which, when executed, generates a plan for dealing with the error. Currently, error recovery involves very little tinkering with the actions that are currently on the timeline, that is, no serious replanning.

Lastly, Phoenix agents have limited abilities to monitor their own progress. This is accomplished by generating expectations of progress, and matching to them actual progress. In the near future, this mechanism (called *envelopes*, Sec. 5.8) will enable Phoenix cognitive components to predict failures before they occur.

In sum, planning is accomplished by adding a selection action to the timeline to search for a plan to address some conditions. Executing the selection action places an appropriate plan action or primitive action on the timeline. If this new entry is a plan action, then when it is executed, it expands into a plan by putting its sub-actions onto the timeline with their temporal inter-relationships. If it is a primitive action, execution instantiates the requisite variables, selects an execution method, and executes it. In general, a cognitive agent will interleave actions from the several plans it is working on.

This style of planning is "lazy skeletal refinement"---lazy because some decisions are deferred until execution time. Specifically, plans are not selected until selection actions are executed, and execution methods are selected only when an action is about to execute. This style of planning and acting is designed to be responsive to a complex dynamic world by postponing decisions, while also grounding potential actions in a framework (a skeletal plan) that accounts for data, temporal and resource interactions. The combination of a reflexive and cognitive component is designed to handle time scale mismatches inherent in an environment that requires micro actions (e.g., following a road) and contemplative processing such as route planning, which involves long search times and integration of disparate data. We must stress, however, that Phoenix is too early in its development to claim that our agent design is *necessarily* the best one for the Phoenix environment (see Sec. 5.8).

5.6. The Organization of Fire-Fighting Agents in Phoenix

The fourth layer of the Phoenix system is the centralized, hierarchical organization of fire fighting agents. Because all agents have the same architecture, many other organizations of agents are possible. Our centralized model is neither robust (e.g., what happens if the fireboss is disabled?) nor particularly sophisticated. But it is simple, a great advantage in these initial phases of the project. One fireboss coordinates all fire fighting agents' activities, sending action directives and receiving status reports, including fire sightings, position updates, and actions completed. The fireboss maintains a global view of the fire situation based on these reports, using it to choose global plans from its plan library. It communicates the actions in these plans to its agents, which then select plans from their own plan libraries to effect the specified actions. Once their plans are set in motion, agents report progress to the fireboss, from which the execution of global plans is monitored. All communication in this centralized implementation is between the fireboss and individual agents - there is no cross-talk among the agents.

The fireboss maintains global coherence, coordinating the available fire fighting resources to effectively control the fire. It is responsible for all the work required to coordinate agents, such as calculating rendezvous points, deciding how to deploy available resources, and noticing when the fire is completely encircled with fireline. The plans in its plan library are indexed by global factors, such as the size of the fire and the weather conditions. The actions in its plans are mostly concerned with coordinating and directing other agents. The fireboss' state memory records the current environmental conditions, where agents have seen fire, what actions have been taken, what agents are available, and how well global plans are progressing. The fireboss is currently implemented without any sensors, effectors, or reflexes. It is a cognitive agent that relies

solely on communication for its knowledge of what develops in the outside world, although it does have a map of the static features of Yellowstone.

Each of the other fire fighting agents has a local view of the environment based on its own sensory input. They have access to maps of the static features in Yellowstone such as ground cover, roads, and rivers, but only know about dynamic processes such as the fire from what they see or are told by the fireboss. Sensors have a limited radius of view, though agents are able to remember what has been perceived but is no longer in view. The fireboss's global view is available to an agent only through communication. A bulldozer is an example of an agent type. It has a movement effector that can follow roads or travel cross-country. When it lowers its blade while moving, it digs fireline and moves more slowly. It has a sensor that sees fire within a radius of 512 meters. Another sensor picks up the contour of a fire (within its radius of view). When a bulldozer is building fireline at the contour, it uses the follow-fire sensor in combination with the movement effector (with lowered blade) and a reflexive action that helps maintain a course parallel to the contour. As the contour changes, the contour sensor registers the change, which triggers a reflex to adjust the movement effector's course. The bulldozer's plan library has plans for simple bulldozer tasks such as following a given path or encircling a fire with fireline.

Although all agents have the same architecture (i.e., timeline, cognitive scheduler, plan library, state memory, sensors, effectors, and reflexes) they do not have the same plans, reflexes, sensors or effectors. The difference between the fireboss and other agents lies in their views of the world and the types of plans each knows. The lines of authority and division of responsibilities are clear; the fireboss maintains the global picture, based on the local views of its agents, and it executes plans whose effects are to gather information, send directives to agents, and coordinate their activity via communications. In contrast, the agents execute plans whose actions program sensors and effectors, which in turn effect physical actions in the world. In some sense the fireboss is a "meta-agent" whose sensors and effectors are other agents.

5.7. An Example

We now return to the example that we introduced in Section 5.2 and used in Section 5.5 to illustrate cognitive scheduling. In this *two bulldozer surround* plan, the fireboss instructs two bulldozers to rendezvous, then go to the fire and build fireline around it in opposite directions. Figures 5.2, 5.3, and 5.4 show the progress of this plan. Each offers two views of the situation. Figure 5.2 shows the real world in the left pane, and the fireboss's view in the right pane. Note that the fireboss is not yet aware of the fire. What it knows about new fires is based on status reports from a watchtower agent (not shown). Each watchtower has a sensor programmed to look for new fires at regular time intervals. When the watchtower spots this fire, it reports the location and size to the fireboss. Based on this report and the resources available, the fireboss selects the two bulldozer surround plan. The first plan steps allocate the bulldozers, which ensures they are not busy with other tasks and assigns them to this plan. The next step instructs them to rendezvous so they can follow the same route to the fire. While they rendezvous, the fireboss locates the rear of the fire (the upwind side), and calculates a route to the fire that approaches it from

that direction. The next two steps communicate to each bulldozer instructions to follow the given path and encircle the fire. They are given clockwise and counterclockwise encircling directions, respectively.

After receiving its instructions, each bulldozer searches its plan library to find a plan for following the path and encircling the fire in the given direction until it closes the fireline. Neither bulldozer knows about the other, nor does either know the full extent or precise location of the fire. Recall that the fireboss doesn't know exactly where the fire is, either, so the path it supplied to the bulldozers may direct them wide of the fire, or, more often, to a location that is burning. In this example, the path given by the fireboss ends in the fire, so the bulldozers will follow the path until they detect it. In Figure 5.3 we see the bulldozers starting to build line. In the fireboss view (right pane) each one appears at the position it had reached when it made its last status report. Thus they are at slightly different positions that are out of date with respect to their real positions in the left hand pane.

When fire is seen, a bulldozer reflex is triggered to stop its movement effector. A cognitive action also notes that the sensor has seen fire and reprograms the sensors and effectors with the right combination of instructions to follow the fire in the direction specified by the fireboss, building fireline as it goes. A message is sent to the fireboss to signal the start of line-building. The bulldozer will continue to build line until instructed to stop by the fireboss. In Figure 5.4 we see in the left pane that the bulldozers have almost encircled the fire. In the right pane is the view of the bulldozer encircling in the clockwise direction. Note that it only knows about the fire it has seen as it was building line. It is just coming within range of the other bulldozer (see the spot of fireline to its northeast).

This simple bulldozer plan, to follow a path and encircle a fire without reference to other bulldozers, can be used by one, two, or many bulldozers. The fireboss, with its global view, picks points around the fire, selects any number of bulldozers, and directs each to go to one of the points and build fireline in a specified direction. The bulldozers act only with regard to their instructions and the local information in their field-of-view. If the bulldozers fail to fully encircle the fire (for whatever reason), the fireboss is responsible for noticing the failure, based on what is reported to it from watchtowers and bulldozers.

Figure 5.10 shows the state of the fireboss's timeline as the bulldozers are closing off the fireline (see Figure 5.4). The network in the top left box is the top level of the timeline, which contains four entries and reads from left to right. There is a startup-action and an end-action (place-holders), and two entries with no temporal constraint between them. The top entry is an action that executes periodically and updates state memory with new information about the environment. The bottom entry is an action that is placed on the fireboss's timeline automatically by the report of a new fire. It causes a plan to be selected from the fireboss's plan library, based on the characteristics of the reported fire, and then expanded on the timeline, as illustrated in Section 5.5. The selected plan is shown in the top right box, and its expansion is shown in the two lower boxes (the plan unfolds left-to-right, and is continued in the lowest box). Entries preceding the shaded one have already been executed. These include allocating each bulldozer, instructing them to rendezvous, calculating a route for them to take to the fire, and (in undetermined order) instructing them to follow that route and encircle the fire.

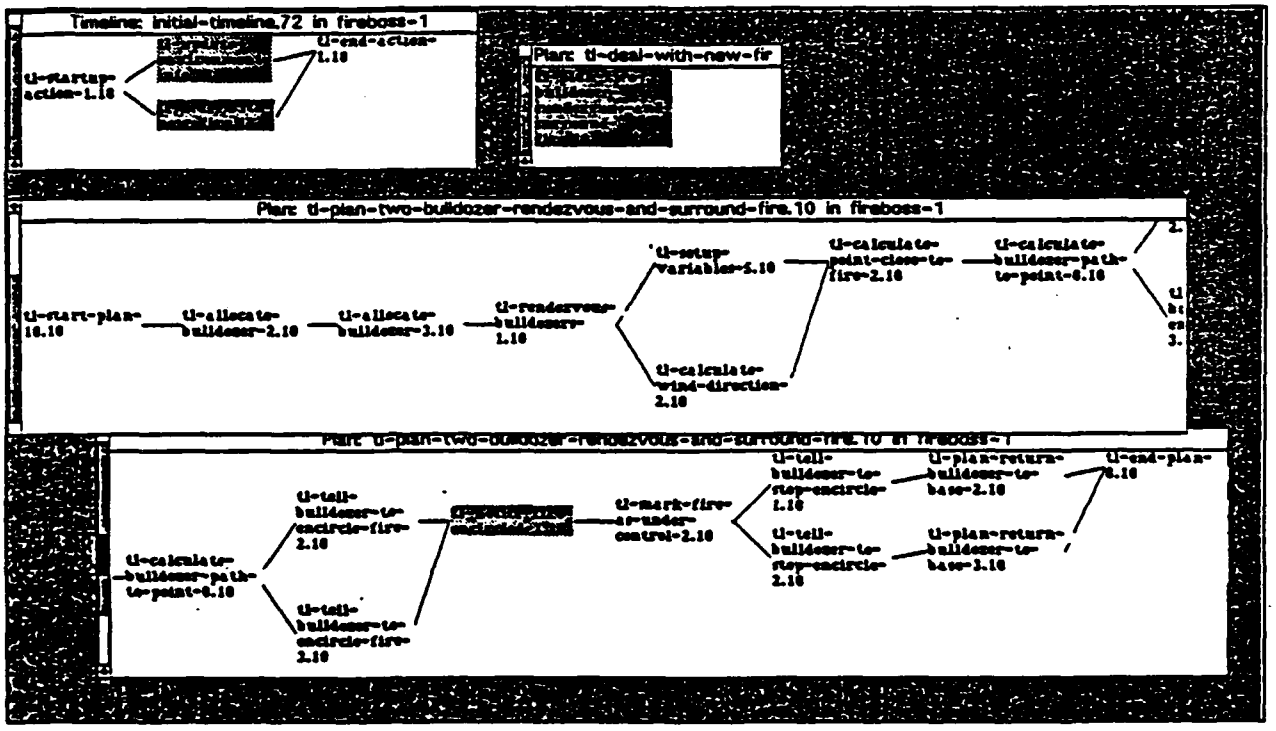


Figure 10: The fireboss's timeline

This figure shows the fireboss's timeline, including plan expansions, as viewed from the Phoenix desktop (headers for boxes name parent nodes). The box at the top left is the top level of the fireboss's timeline. The darkly shaded entry here is the action to deal with a newly sighted fire. Dark shading means an entry is executing or has a child node that is executing. The fireboss selected the two-bulldozer-rendezvous-and-surround-fire *plan, which is shown in the top right box. The expansion of this plan is shown in the lower boxes. It starts with the timeline entry* tl-start-plan-10.10, *reads from left to right and wraps around into the lower box. Entries are ordered in temporal sequence; a split after an entry represents subsequent entries with no temporal constraints between them. The current entry (shaded) is executed by periodically checking to see whether the fire is completely encircled.*

The fireboss is currently waiting for the bulldozers to finish encircling the fire (see shaded entry in lowest box). The execution method for this entry runs periodically, checking to see if there is continuous fireline surrounding the fire. Once this is true, the bulldozers will be instructed to stop building fireline. This is necessary because the fireboss maintains the global view of the fire, and must tell the bulldozers when the fire is surrounded. (The bulldozers' timelines are not shown.). Once each bulldozer has been instructed to stop building line, a plan will be selected for returning it to the bulldozer base.

5.8. Current Status and Future Work

The Phoenix system is very much a work in progress. As is clear from the preceding sections, several important aspects are handled in a rudimentary or preliminary way. Currently five people are pursuing research and enhancing the system in the areas described below:

More sophisticated plans: We have about a dozen plans that attack fires directly, with up to four bulldozers, building line at the fire front until the fire is encircled. We are starting to develop *indirect attack* plans that incorporate natural barriers. This requires more knowledge and coordination on the part of the fireboss; since other agents can't see the fire unless they are close to it, the fireboss must guide their activities when they are working at a distance. The fireboss must take advantage of natural barriers when deciding where to build fireline, which requires the ability to project the spread of the fire and the progress of fire fighting agents. As we develop new and more sophisticated plans, we must also enhance the mechanisms by which agents select plans. Currently, the keys for selecting plans are just wind speed and the availability of fire-fighting agents; as well as some plan-specific criteria such as whether bulldozers are nearby or distant when the plan is selected. The keys will have to become more discriminating, and we will probably have to develop more sophisticated plan-selection mechanisms.

Monitoring: We have designed a general monitoring mechanism called *envelopes* that minimizes the cognitive resources devoted to monitoring while providing early warning of plan failure. Envelopes incorporate expectations of how plans are to proceed; they represent these expectations functionally. As actual progress is reported, it is compared with these expectations, and deviations outside certain parameterized thresholds are flagged for cognitive attention. For example, if an agent must be at a certain place at a certain time, we can tell by projection whether the deadline is feasible - can the agent travel the distance in the given time? By projecting the expected time of travel (based on a parameter such as average speed for the agent on the given terrain), we can create an envelope for the travel time, and use it to monitor the agent's progress. The envelope also predicts the expected arrival time, based on the recent progress of the agent. Furthermore, it predicts the minimum speed at which the agent must travel over the remaining distance to arrive before the deadline. If this speed is at or approaching the top speed of the agent, then the envelop signals the planner that the deadline is in jeopardy,

providing an early warning of failure. Currently, we have hooks for envelopes in plans, but we do not have the mechanisms to replan when envelopes are violated.

Error recovery and replanning: These activities are implemented as cognitive actions, just like plan selection and plan expansion. When an error is detected in a plan, an action is posted to the timeline that inspects the error and attempts to fix the existing plan. Consider, for example, a failure on the part of a bulldozer to find fire at the location to which it was sent. A plan we currently have to fix the error is to travel a little further in a specified direction, looking for the fire. A really intelligent error recovery will know when to try cheap fixes, such as modifying a destination; and when to begin a search for a way to significantly modify a plan (e.g., by dispatching another bulldozer); and when, as a last resort, to abandon the current plan and begin from scratch. Error recovery and replanning will depend significantly on intelligent monitoring; in fact, envelopes are designed to predict errors before they happen, minimizing "downtime."

Cognitive scheduling: We need to enhance the scheduling abilities of the cognitive component to make agents responsive to real-time demands in fire fighting. This is particularly true for the fireboss in our implementation, since it is essentially a cognitive agent. Currently, scheduling involves three actions: selecting an action to execute, deciding how much time is available, and selecting an execution method. But although these actions are "charged" for the time they use, they are not themselves scheduled, nor are there multiple execution methods to implement them. In short, the cognitive scheduler is a separate "interpreter" of the timeline. To make the scheduling of actions completely uniform, scheduling actions must themselves be scheduled. In addition, we must develop scheduling strategies, along the lines suggested in Lesser, Durfee, and Pavlin's approximate processing proposal [Lesser et al., 1988].

Agent Architecture. To facilitate experiments with different agent designs in different environments, we have started to build a generic agent architecture. It is a collection of parameterizable structures that represent the design of parts of an agent. For example, our generic action structure includes pointers to execution methods, to envelopes, and to predicates that are tested before the action is selected. Generic execution methods, in turn, contain estimates of their time requirements, their prerequisites, and on. We also have generic structures for sensors and effectors. In the near future, we will implement generic structures for strategies, including memory access strategies and cognitive scheduling strategies. The eventual goal is a full generic agent architecture that makes it easy to implement different agent designs by specifying how the agent manages its sensors and effectors, how it manages its memory, and how it decides what to do next.

Organization and communication: We have demonstrated one way to organize a multi-agent planner in the Phoenix testbed, but the agent architecture certainly supports others. Work is underway in Victor Lesser's lab to build a cooperating, distributed planner for the Phoenix testbed. Although preliminary, this model assumes multiple firebosses, each with *spheres of influence* (geographic areas and agents) under its control, who cooperatively fight fires at their borders, loaning resources to neighbors, or redrawing their boundaries to shift the work load in

times of stress. While this model is similar to the Phoenix planner in the relationship between firebosses and agents, it adds a cooperative relationship between firebosses.

Learning: Phoenix agents should learn to improve their performance. The opportunities for learning are myriad: we can learn new reflexes, and chain reflexes together to learn short plan fragments. We can learn new plans from patches to failed ones. We can learn correlations between environmental conditions, such as changes in wind direction, and failures, such as bulldozers becoming trapped in the fire. Currently, we are extending the error recovery mechanisms to learn patches to failed plans. This is one aspect of Adele Howe's dissertation work [Howe, 1989] Allen Newell recently pointed out that "you can't program SOAR" because much of its behavior emerges from sequences of locally-selected chunks, and there is really no way to predict how a chunk, added by hand, will make the system behave. We have found the same to be true of actions and reflexes in Phoenix, and concurr with Newell that once a system attains a degree of complexity, it must learn to improve its performance itself.

5.9. Conclusion.

The development of Phoenix has been intimately tied to our evolving ideas about AI research methodology, and specifically to our understanding of the role of evaluation in AI research [Cohen and Howe 1988a, 1988b] . Clearly, the evaluation of Phoenix must be with respect to the goals of the project. Moreover, it must tell us not only whether we have succeeded, but whether we are succeeding; and why, or why not. The goals of Phoenix are, as noted in Section 5.1, of three kinds. Our technical goals are to build a real-time planner with learning, approximate scheduling, envelopes, and the other features noted above. Our scientific goal is to understand how environmental characteristics influence agent design---the relationships discussed in the context of the behavioral ecology triangle (Fig. 5.5). Lastly, we are using Phoenix as a framework in which to develop AI methodolgy.

Progress toward each of these goals is evaluated differently. Phoenix is parameterized and instrumented at all its layers to facilitate evaluations of specific technical developments; for example, we can assess whether an approximate scheduling algorithm is robust against varying time pressure because we can vary time pressure while holding other factors constant. We can run fire scenarios in dozens of conditions, with dozens of variations in the algorithms used by the Phoenix planner. These experiments are scheduled to begin in the Fall of 1989. They will enable us to demonstrate the utility of our technical solutions, explain why they are solutions, and discover the limits on their scope [Cohen and Howe, 1988a].

But clearly, these cannot be the only aims of the experiments. While it is valuable to probe the scope and efficacy of specific techniques, such experiments will not necessarily address our scientific goals. We might show that a Phoenix planner works well in the Phoenix environment, but not how the environment constrains the design of planners. Furthermore, unless we are trying to answer specific questions of this sort, experiments with techniques will be unguided. There are dozens of variations on the Phoenix planner,

and hundreds of environmental conditions in which they might be tested. To guide the search of this space, we will generate and test general rules that justify and explain the design of agents. These rules will call upon functional relationships that capture tradeoffs. For example, the well known store-recompute tradeoff lurks in the design of the Phoenix planner: we use it to justify the decision to rely on stored plans in an environment that exerts time pressure, favoring storage over computation. Perhaps there is a general rule here (e.g., under time pressure, rely on storage over computation), or perhaps there are many specific variants of this rule, for environments with different kinds of time pressures and agents with different kinds of store-recompute tradeoffs. In any case, our scientific goal is to discover functional relationships (and to exploit those we already know, like the store-recompute tradeoff), and to embed them in rules for designing intelligent agents. To evaluate progress, we need to measure not the performance of the agents, but the extent to which that performance can be predicted. If we really understand the relationships between environment characteristics, agents' behaviors, and agents' designs, then we should be able to predict that agents with particular designs will behave in particular ways under particular environmental conditions.

Although we are far from this goal, it is paradigmatic of the style of AI research we advocate. To evaluate the success of this methodological stance will take a long time, but if it is possible, there is surely no better aim for AI than to understand---to the point of being able to predict behavior---how to design intelligent agents in complex environments.

Part IV

Plausible Reasoning

Chapter 6

Beyond ISA: Structures for Plausible Inference in Semantic Networks

6.1 Introduction

Can cough syrup make people drunk? Our favorite brand can, because it contains alcohol. If you didn't already know that cough syrup is intoxicating, you could infer it from two specific propositions—cough syrup contains alcohol and alcohol is intoxicating—and from a general plausible inference rule:

Rule 1

$$\frac{\begin{array}{lll} x & \text{CONTAINS} & y, \text{ and} \\ y & \text{CAUSES} & z \end{array}}{\begin{array}{lll} x & \text{CAUSES} & z \end{array}}$$

Other familiar rules of plausible inference include property inheritance (e.g., cats have five toes, Ginger is a cat, so Ginger has five toes) and causal abduction (e.g., fires cause smoke, so if you see smoke, look for a fire).

Rules like these have two roles that we expect to become increasingly important in coming years. First, they support *graceful degradation* of performance at the boundaries of our knowledge. A *brittle* knowledge system that doesn't know explicitly whether cough syrup makes you drunk won't offer a plausible answer—it simply won't answer the question [Lenat *et al.*, 1986], [Lenat and Feigenbaum, 1987], [Collins *et al.*, 1975]. Graceful degradation depends on general knowledge, which we formulate as as plausible inference

rules such as Rule 1, to make up for a lack of specific knowledge. Second, we expect plausible inference to reduce the effort of building knowledge bases, because knowledge engineers needn't state explicitly those propositions that can be plausibly inferred. Property inheritance, for example, relieves us from having to state explicitly that each member of a class has each property of that class [Brachman and Schmolze, 1985]. Rules like property inheritance and Rule 1 obviously are needed to build "mega-frame" knowledge bases [Lenat and Feigenbaum, 1987].

Rule 1 has the same structure as property inheritance over ISA links, and can serve the same purposes, that is, supporting graceful degradation and knowledge engineering. We have developed a simple method for deriving such rules from the relations in a knowledge base, and we have shown how to differentiate plausible ones from implausible ones based on their underlying "deep structure."

This paper describes two empirical studies of these rules. Both depend on a moderately large knowledge base that we developed for the GRANT project [Cohen *et al.*, 1985], [Cohen and Kjeldsen, 1987], [Kjeldsen and Cohen, 1987]. The GRANT KB contains roughly 4500 nodes linked by 9 relations and their inverses. In the first study we derived approximately 300 plausible inference rules from these relations. Then we generated over 3000 specific inferences by replacing the variables in the rules with concepts from the GRANT KB, and presented them to human subjects to discover which syntactically permissible rules were plausible (Sec. 6.2). The second study tested the hypothesis that the plausibility of these rules can be predicted by whether they obey a kind of transitivity (Sec. 6.2.5). We will begin by describing these studies, hypotheses, and results. Then we will discuss the role of knowledge in assessing the plausibility of inferences.

6.2 Experiment 1: Identifying Plausible Rules

In this section we describe how to use the structure of property inheritance to produce many other plausible inference rules, and how we determined the plausibility of these rules.

6.2.1 Background

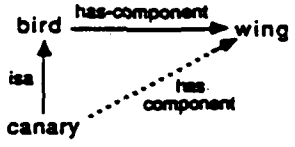
Property inheritance over ISA links can be written

$$\begin{array}{lll} n_1 & \text{ISA} & n_2, \\ n_2 & \text{R} & n_3 \\ \hline n_1 & \text{R} & n_3 \end{array} \quad \text{and}$$

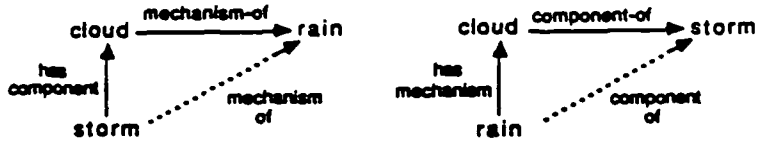
where the relation R between n_2 and n_3 is viewed as a property of n_2. For example, if a canary is a bird and bird HAS-COMPONENT wings, then canary HAS-COMPONENT wings (Fig. 6.1.a). Here, R is HAS-COMPONENT and the inherited property is "HAS-COMPONENT wings." Many plausible inference rules have this structure, but inherit over links other than ISA. For example, in the "cough syrup" inference, above, cough syrup inherits the "CAUSES intoxication" property over the CONTAINS relation:

cough syrup	HAS-COMPONENT	alcohol,	and
alcohol	CAUSES	intoxication	
cough syrup	CAUSES	intoxication	

Figure 6.1.b shows two other examples. They have the same premises but different conclusions. One premise is "storm HAS-COMPONENT cloud" (and, equivalently, "cloud COMPONENT-OF storm"); the other is "cloud MECHANISM-OF rain" (and, equivalently, "rain HAS-MECHANISM cloud"). But the conclusions are "storm MECHANISM-OF rain" and "rain COMPONENT-OF storm," respectively.



a. Property inheritance over ISA links



b. Inferences that have the structure of property inheritance

Figure 6.1: Inference from property inheritance and structurally-identical rules

This illustrates that each pair of relations can produce two plausible inference rules that have the same structure as property inheritance over ISA links. For relations R1, R2 these rules are:

$$\textbf{Rule 2}\quad \frac{\begin{array}{lll} n_1 & \text{R1} & n_2, \\ n_2 & \text{R2} & n_3 \end{array}}{\begin{array}{lll} n_1 & \text{R2} & n_3 \end{array}} \quad \text{and}$$

and

$$\textbf{Rule 3}\quad \frac{\begin{array}{lll} n_3 & \text{R2-INV} & n_2, \\ n_2 & \text{R1-INV} & n_1 \end{array}}{\begin{array}{lll} n_3 & \text{R1-INV} & n_1 \end{array}} \quad \text{and}$$

Figure 6.1.b shows these alternatives for R1 = HAS-COMPONENT, R2 = MECHANISM-OF, n_1 = storm, n_2 = cloud, and n_3 = rain.

Figure 6.1 introduces the notation we will use throughout. Rules are represented as triangles formed from three concepts and three relations. The legs of the triangle represent premises, and are always drawn as solid lines. The hypotenuse represents the conclusion and is always drawn as a dashed line.

Rules can be chained by letting the conclusion of one serve as a premise for another. Figure 6.2 shows how the conclusion of a *first generation* inference, "storm MECHANISM-OF rain," serves as the premise of a *second generation* inference, which has the conclusion "storm HAS-PRODUCT runoff."

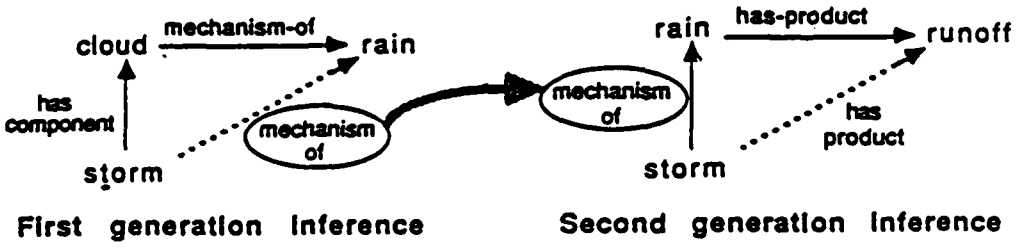


Figure 6.2: Second-generation inference

Since each pair of relations produces two rules, a knowledge base constructed from N relations will produce $(N^2 + N)/2$ pairs of relations (including relations paired with themselves) and an equal number of rules. The GRANT KB is constructed from nine relations and their inverses, so $(18^2 + 18)/2 = 342$ were generated.

Experiment 1 had two goals. One was to generate all possible rules for the GRANT KB and to determine which of them produce plausible conclusions. The other was to find out how the plausibility of conclusions is affected

by chaining these rules. Applying roughly 300 rules to the GRANT KB (as we describe below), produced thousands of first generation inferences and over 200,000 second-generation inferences. We expected very few of these to be plausible; but, if we could discover or predict the plausible ones, then we would have a powerful method to reduce the effort of constructing large knowledge bases.

6.2.2 Design

To determine whether the rules produce plausible conclusions, we first *instantiate* them with specific concepts, then present them to human subjects to judge.

We derived 315 rules from the GRANT KB.[1] For each we produced 10 *test items* (five first generation items and five second generation items) by the following method:

Each rule is based on two relations. For each pair, say HAS-COMPONENT and MECHANISM-OF, we search the GRANT KB for triples of nodes n_1,n_2,n_3 that are connected by these relations (i.e., n_1 is connected to n_2 by HAS-COMPONENT, and n_2 is connected to n_3 by MECHANISM-OF). Each triple represents a pair of premises from which *two* inferences can be drawn (see Rules 2 and 3, above). For instance, storm, cloud, and rain instantiate n_1, n_2, and n_3, respectively in Figure 6.1.b, yielding the conclusions "storm MECHANISM-OF rain" and "rain COMPONENT-OF storm."

Most pairs of relations in the GRANT KB yield dozens of n_1,n_2,n_3 triples. We randomly select five, and their conclusions, to be first generation test items. However, we add the conclusions of *all* the triples to the GRANT KB.

This procedure is repeated to generate second generation test items, with the added condition that one premise of each second generation item must be a conclusion that was produced during the previous search (though not necessarily the conclusion of a first generation test item).

In all, the 315 rules yield a data set of 3116 test items, of which roughly half are first generation and half are second generation items.[2]

6.2.3 Procedure

Items in the data set were presented to human subjects by a computer program. Subjects were asked first to indicate whether both premises were

[1] Pruning duplicates reduces the original 342 rules to 315.

[2] We don't have 3150 items because, for some rules, the GRANT KB yielded fewer than five first generation instances.

acceptable, one or both were unacceptable, or they did not understand one or both premises. Next, the conclusion was shown and subjects were asked to judge whether it followed or did not follow from the premises, or else to indicate that they did not understand the conclusion. Each item was seen by two subjects. Following a practice session with 20 items (none of which was in the data set), each subject judged approximately 700 items from the data set. This took about five hours, distributed over three or four self-paced sessions.

6.2.4 Results

Since the premises of the test items came from an existing knowledge base we expected that most would be judged acceptable. This is in fact the case: 82% percent of first generation premises and 63% of second generation premises were judged to be acceptable. The following results pertain only to those items.

Each rule is represented in the data set by five first generation items and five second generation items, and each item was seen by two subjects. Thus, 10 judgments are made of the items in each generation of each rule. Two *plausibility scores* for a rule, ranging from 0 to 10, are equal to the sum of the number of items that subjects judged plausible for each generation of each rule. The mean plausibility score, over the 315 rules, for first generation items is 4.18 (var. = 6.92), and the corresponding statistic for second generation items is 3.17 (var. = 4.88). Both are significantly different from chance and from each other at the $p < .01$ level. The fact that both are *below* chance means that the preponderance of rules are not plausible. Given this, one would expect chaining of inferences to produce increasingly-implausible conclusions. This is supported by the evidence that second generation inferences are significantly less plausible than first generation ones. Subjects judged approximately 50% of the rules to have plausibility scores between 3 and 7 (of a possible 10); they judged the rest of the rules to be predominantly plausible or implausible.

6.2.5 Discussion

While these results indicate that many rules generate predominantly plausible conclusions, and many others are predominantly implausible, they do not tell us how to predict which will be plausible and which will not. We wanted to find a small set of common characteristics of rules on which to base these predictions. Furthermore, we wanted these characteristics to depend

only on the relations in the rules, not on the nodes or any other exogenous factors.

We discovered two common aspects of relations. Some relations, such as HAS-COMPONENT have a *hierarchical* interpretation. Others, such as CAUSES, can be interpreted as *temporal* relations. Lastly, relations such as MECHANISM-OF can have both hierarchical and temporal interpretations: in "n_1 MECHANISM-OF n_2," n_2 may be a process that hierarchically subsumes the mechanism n_1, or n_1 may be an object or process that exists or is required prior to achieving n_2. Table 6.1 lists the *deep relations* that correspond to all 18 *surface* relations. Each deep relation has a h (hierarchical) or t (temporal) interpretation, or both. Expressing rules in terms of

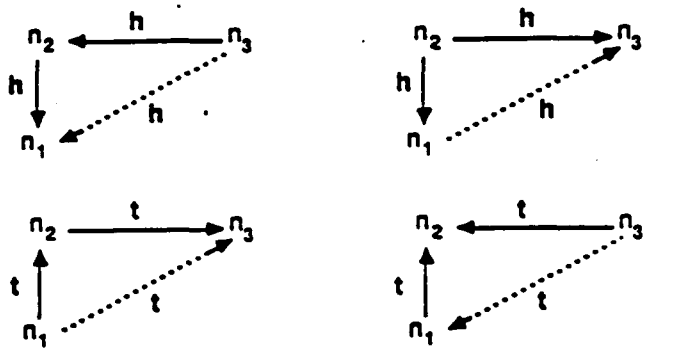
Surface relation	Deep structure	Surface relation	Deep structure
CAUSES	t →	CAUSED-BY	← t
COMPONENT-OF	← h	HAS-COMPONENT	h →
FOCUS-OF	← h	HAS-FOCUS	h →
MECHANISM-OF	t → ← h	HAS-MECHANISM	← t h →
PRODUCT-OF	← t	HAS-PRODUCT	t →
PURPOSE-OF	h ← t → t	HAS-PURPOSE	h → t ← t
SETTING-OF	h →	SETTING	← h
SUBJECT-OF	← h	SUBJECT	h →
SUBFIELD-OF	← h	HAS-SUBFIELD	h →

Table 6.1: Surface relations and corresponding deep relations

these deep relations reduces the set of 315 surface rules to 95 unique *deep structures.*

More importantly, we identified a characteristic of deep structures, called *transitivity*, which seemed to explain why some rules were plausible and others implausible. Figure 6.3 shows two transitive structures and two intransitive ones. The transitive deep structures represent the rules: "If n_1 CAUSES n_2, and n_2 CAUSES n_3, then n_1 CAUSES n_3," and "If n_1 COMPONENT-OF n_2, and n_2 COMPONENT-OF n_3, then n_1 COMPONENT-OF n_2." We call these structures transitive because the premises imply an ordering between n_1 and n_3 that, to be preserved, *requires* a particular ordering between n_1 and

n_3 in the conclusion (n_1 to n_3 in one rule and n_3 to n_1 in the other). In contrast, the intransitive structures do not require any ordering on nodes in the conclusion. In one, the premises indicate no hierarchical ordering between n_1 and n_3, only that n_2 is hierarchically-superior to both. Similarly, in the other intransitive rule, n_1 and n_3 are both temporally-prior to n_2, but no ordering is implied between them and, thus, required in the conclusion.[3] The mean plausibility score for transitive rules was 8.94 (out



a. Example transitive deep structures

b. Example intransitive deep structures

Figure 6.3: Transitive and intransitive deep structures

of 20; var. = 16.83), and for intransitive rules, 5.89 (var. = 14.46). Again, the preponderance of these rules are judged implausible, but these values are significantly different ($p < .01$), and provide strong post-hoc evidence that transitivity is a factor.

Transitivity is clear when surface relations map to deep relations whose h and t elements point in just one direction. But the surface relations HAS-MECHANISM and PURPOSE-OF have deep relations where t and h point in opposite directions. Therefore, rules that are transitive under one interpretation of these relations are necessarily intransitive under the other. For example, the structure in Figure 6.4.a may be transitive or intransitive. We call structures like this *ambiguous*.

Although our data suggested that transitivity predicts the plausibility of rules with unambiguous structures, the results were less clear for ambiguous ones. All ambiguous structures have transitive interpretations, but we knew

[3]The term transitivity refers to the form of the deep structure, and does not imply mathematical transitivity.

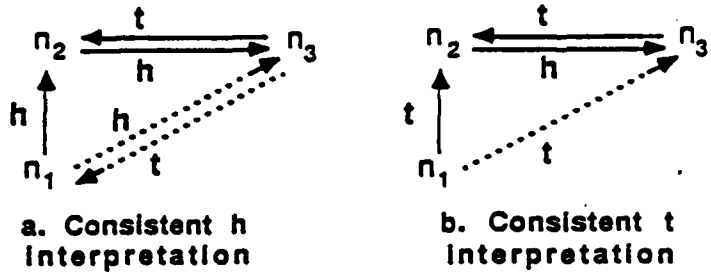


Figure 6.4: Ambiguous deep structures

from our data that not all the corresponding rules were plausible. We hypothesized a characteristic of interpretations, called *consistency*, that might discriminate plausible ambiguous rules from implausible ones. A structure has a consistent interpretation when its deep relations all have the same interpretation, either h or t. For example, Figure 6.4.a has a consistent interpretation in which all its deep links can be interpreted as h. Moreover, this h interpretation is transitive. Figure 6.4.b has a consistent t interpretation, but it is intransitive; and the interpretations of the deep relations that make Figure 6.4.b transitive are inconsistent (t, t, and h).

6.3 Experiment 2: Plausible inference as transitivity

At the end of Experiment 1, we had formed the hypotheses that transitivity predicts plausibility, and that consistency determines the interpretation (transitive or intransitive) of ambiguous structures. Experiment 2 tests these hypotheses.

6.3.1 Design

Experiment 2 focused on ten relations from Experiment 1: CAUSES, COMPONENT-OF, MECHANISM-OF, PRODUCT-OF, PURPOSE-OF and their inverses. (The other relations replicate deep relations and occurred relatively infrequently in the knowledge base.) Since each of these surface relations has a unique corresponding deep relation, the 95 rules they generate map to 95 different deep structures. From these, we chose 56 structures (and

thus, rules) as a representative sample.[4] We generated 10 first generation test items for each of the 56 rules, following the procedure described in Experiment 1.

6.3.2 Procedure

Fourteen subjects each viewed all the test items. Items were presented as in Experiment 1.

6.3.3 Results

Our hypothesis is that transitivity, as determined by the consistent interpretation of the deep structure, predicts plausibility. Eight rules are composed of surface relations that have just one deep interpretation (CAUSES, CAUSED-BY, HAS-COMPONENT, COMPONENT-OF; see Fig. 6.5). With these we can analyze the effects of transitivity and consistency on plausibility in rules with single interpretations. A two-way analysis of variance found a

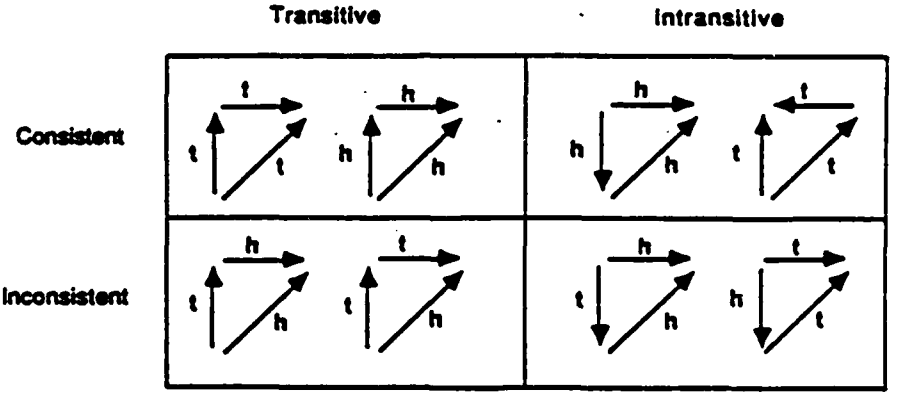


Figure 6.5: Single interpretation deep structures

significant main effect of transitivity ($p < .001$) and a significant transitivity × consistency interaction ($p < .001$), but no main effect of consistency ($p > .2$), confirming that transitivity predicts the plausibility of these rules. A graph of the means (Fig. 6.6) suggests that we cannot predict the plau-

[4]Rules generated from a single surface relation and its inverse always map to one transitive and two intransitive deep structures. Our sample included the transitive structure and one of the intransitive structures (chosen randomly). Pairs of non-identical relations and their inverses form four transitive and four intransitive rules. Our sample included two transitive and two intransitive rules from each of these sets.

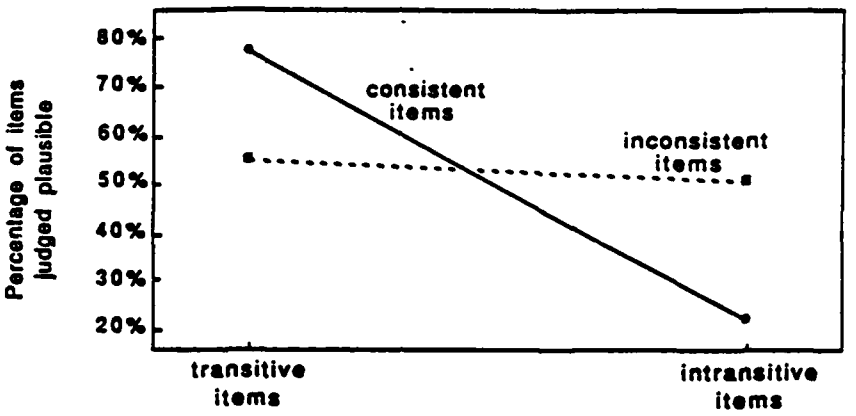


Figure 6.6: Transitivity × consistency analysis

sibility of rules that have no consistent interpretation, because the mean plausibility score for these rules is roughly five out of 10 (i.e., at chance) irrespective of whether the rule is transitive. Figure 6.7 compares the mean plausibility scores of transitive, intransitive, and inconsistent rules to chance performance; transitive and intransitive inconsistent items are collapsed into one category.

Analyzing all our rules in terms of these categories yields 18 that have consistent transitive interpretations, 20 consistent intransitive rules, 8 inconsistent rules, and 4 rules that have both transitive and intransitive consistent interpretations.[5] The histogram for all rules (including the eight analyzed earlier) is presented in Figure 6.8.

Although less clear-cut, Figure 6.8 echoes one of our earlier results: transitivity predicts the plausibility of rules with consistent interpretations. However, the mean plausibility score for inconsistent rules is higher than chance, and the mean plausibility score of consistent intransitive rules is much closer to chance than it was in Figure 6.7.

6.3.4 Discussion

While the predictive power of transitivity is high for rules that have only one interpretation, it becomes diluted in rules with multiple interpretations. It is not surprising that rules with consistent transitive *and* intransitive

[5]Unfortunately, the test items for the other six rules shared many common premises. This was an unavoidable consequence of our decision to generate test items randomly. Four had consistent transitive interpretations, two had consistent intransitive interpretations.

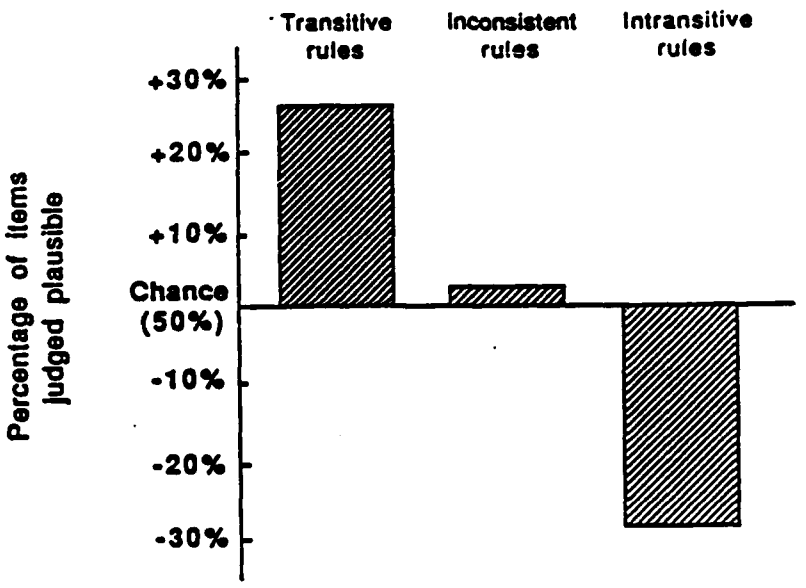


Figure 6.7: Plausibility scores for rules with single interpretations, expressed as deviations from chance performance

interpretations have a mean plausibility score roughly halfway between the scores for transitive and intransitive rules (Fig. 6.8). However, the mean plausibility score of inconsistent rules, which we expected to be at chance, was higher (61%); and the mean plausibility score of rules with consistent intransitive interpretations, which we expected to be implausible, was not as low as we expected (43%).

We hypothesize that both these effects are due to an unanticipated factor that is raising the plausibility of some but not all of these rules. Whereas all our surface rules have the same structure as property inheritance over ISA links, some but not all of the *deep structures* of both the intransitive and inconsistent rules have this form. For example, the deep structure for the rule n_1 COMPONENT-OF n_2, n_2 HAS-MECHANISM $n_3 \rightarrow n_1$ HAS-MECHANISM n_3 is intransitive, but its conclusion is often plausible, as illustrated in Figure 6.9. In this instantiation, battle inherits "HAS-MECHANISM weapon" from war over a COMPONENT-OF relation. We expect rules with this structure to yield relatively high plausibility ratings even if they are intransitive, because property inheritance is a common and powerful plausible inference rule.

Generalized property inheritance (GPI) is a characteristic of a rule's deep structure, comparable with transitivity:

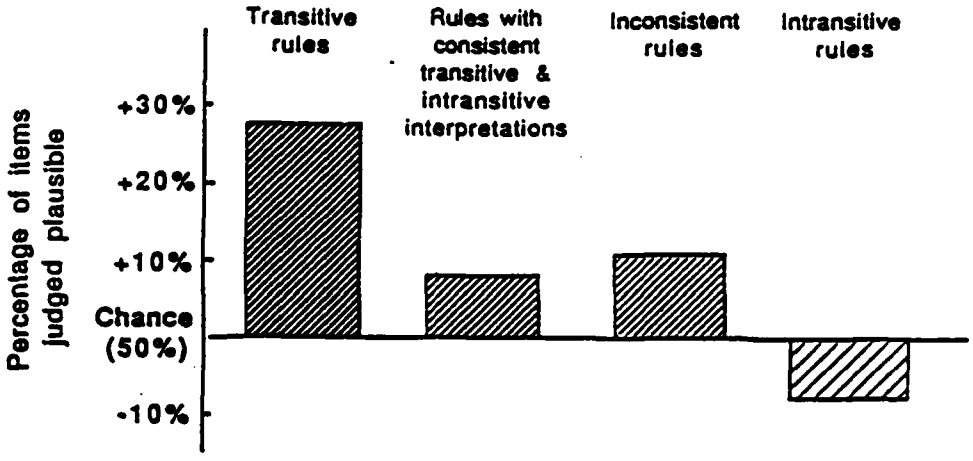
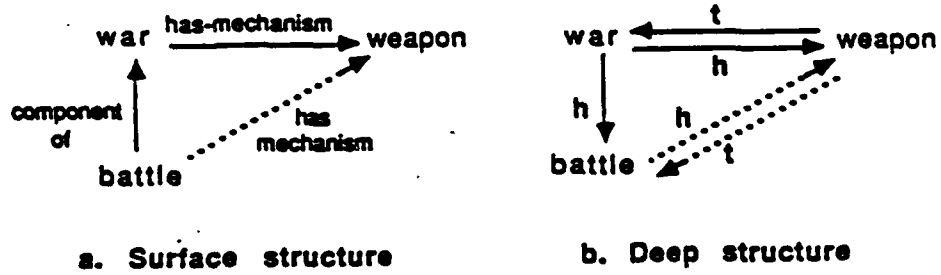


Figure 6.8: Plausibility scores for all rules, expressed as deviations from chance performance



a. Surface structure b. Deep structure

Figure 6.9: Surface and deep structures of an intransitive but plausible rule

> If n_1 is related to n_2 by h, and n_2 is related to n_3 by any relation i, then it is plausible to infer that n_1 is related to n_3 by i

This definition does not restrict the direction of h; it can point "up" or "down" from n_1 to n_2, whereas in property inheritance over ISA links, n_1 must be a subclass or instance of n_2, that is, ISA must point "up." We relax this for GPI because it is often plausible to infer that a concept will have properties of those concepts hierarchically-inferior to it.

GPI explains why some intransitive rules have higher than expected plausibility scores. Since some transitive rules are also GPI, we ran a post-hoc transitivity × GPI analysis of variance, and found main effects of transitiv-

ity ($p < .001$) and GPI ($p < .05$), with no interaction effect. Post-hoc tests on the means (Newman-Keuls) found a significant difference between GPI intransitive items and non-GPI intransitive items ($p < .05$), which means that among intransitive rules, GPI differentiates two statistically-distinct classes—relatively plausible and relatively implausible rules. After removing GPI rules, the mean plausibility score of inconsistent rules decreases (Fig. 6.10). Therefore, GPI provides a post-hoc explanation of why intransitive and inconsistent rules have higher-than-expected plausibility scores. Among transitive items, GPI had no statistically discernible effect. And since there was no interaction between transitivity and GPI, we regard them as independent factors.

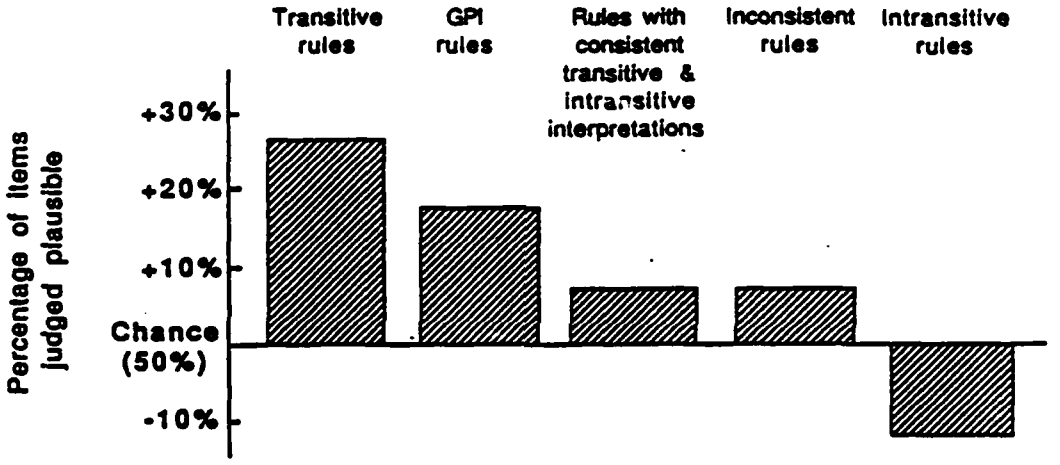


Figure 6.10: Post-hoc revision of Figure 6.8, treating GPI rules separately

6.4 General Discussion—Judging plausibility

In this section we will discuss the factors that contribute to judgments of plausibility. Recall that our goal is to find plausible inference rules that support graceful degradation and help knowledge engineers. Ideally, the agent who uses these rules should not need much knowledge to judge the plausibility of their conclusions. For example, the plausibility of the conclusion of the rule

$$\begin{array}{llll} n_1 & \text{CAUSES} & n_2, & \text{and} \\ n_3 & \text{CONTAINS} & n_1 & \\ \hline n_3 & \text{CAUSES} & n_2 & \end{array}$$

seems not to depend on the objects that instantiate n_1, n_2, and n_3. In contrast, to judge the plausibility of a conclusion of the rule

$$\frac{\begin{array}{l} n_1 \quad \text{CAUSES} \quad n_2, \quad \text{and} \\ n_2 \end{array}}{n_1}$$

we need knowledge about n_1 and n_2 that can tell us how likely n_1 is given n_2.

Several authors have noted the tradeoff between the amount one knows about a plausible inference and and one's confidence in its conclusion [Collins *et al.*, 1975], [Collins, 1978], [Baker and Burstein, 1987], [Polya, 1954]; for example, Polya notes that "In order to judge the weight of the evidence, you have to be familiar with the domain; in order to judge the weight with assurance, you have to be an expert in the domain." [Polya, 1954, p. 114].

What knowledge contributes to the plausibility of the conclusions of the rules in Experiments 1 and 2? Said differently, what factors account for the *total variance in judgments of plausibility* (T) among our subjects? We believe T has four additive components:

subject variance—the proportion of T due only to individual differences in subjects' knowledge, experience, motivation, and so on.

item variance—the proportion of T due only to differences in the concepts that instantiate n_1, n_2, n_3 in the rule.

between-rule variance—the proportion of T due only to differences in the surface structures of rules.

deep structure variance—the proportion of T due only to whether deep structures are transitive, intransitive, or GPI structures.

Ideally, deep structure variance should account for the largest component of T. If 100% of T was due to deep structure variance, then transitivity and GPI would be perfect predictors of plausibility. In contrast, if a large fraction of T is due to item variance, then one needs to know the specific instantiation of a rule—the concepts in the test item—to predict its plausibility. Similarly, between-rule variance represents the effect of knowing the surface structure of test items on one's ability to predict their plausibility. Subject variance represents the limit of our ability to predict plausibility.

For transitive and intransitive rules, and to a lesser extent for GPI rules, deep structure variance accounts for a large fraction of T. For all test items with these structural characteristics, our predictions of plausibility will be correct for 77% of transitive items and 68% of GPI items; and our prediction of *im*plausibility will be correct for 62% of intransitive items. Since these numbers are not 100%, the remaining variance in T must be due to the rule, item, and subject factors.

We estimated between-rule variance as follows: We ran three one-way analyses of variance, by rule, for transitive, intransitive, and GPI rules. This allowed us to make a rough estimate of the proportion of the variance in each of these categories due to rule (as suggested by [Hays, 1973, p. 485]). Between-rule variance is 16% for transitive rules, 27% for intransitive rules, and 52% for GPI rules. That is, if a rule is transitive, then knowing *which* rule it is provides little additional information about the plausibility of items. However, this knowledge accounts for much of the variance in plausibility scores of intransitive and GPI items.

Given that we know the specific rule, how much of the remaining variance (the *within-rule* variance) is due to the individual item and how much is due to subject differences? When the within-rule variance is low, all the items in the rule received approximately the same plausibility score. Therefore, if you know that an item is an instantiation of one of these rules, knowing *which* instantiation it is does not help you predict its plausibility: Most of the within-rule variance is due to subjects. But when the within-rule variance is not low, it may be due either to item, subjects, or a combination of both. One way a rule can have a high within-rule variance is if the plausibility scores of the items fall at both extremes of the scale. For example, if half the items were judged plausible by all subjects and the other half were judged implausible by all subjects, then none of the variance is due to subjects and all is due to items. On the other hand, if many items received split plausibility scores (i.e., half the subjects found them plausible, the other half did not) then much or all of the remaining variance is due to subjects. Thus, we can use the number of split plausibility scores as a rough estimate of the variance due to subjects.

Twenty-two of our 50 rules had low within-rule variance. (The distribution of within-rule variances was skewed low, but ranged from 0 to 19 with a mean of 6.5; a variance ≤ 5.0 was "low.") For these rules, subject differences seem to contribute more to the within-rule variance than do item differences. That is, knowing the instantiation of these rules does not improve our predictions of their plausibility over the prediction we can make from structure

and rule knowledge. The remaining 28 rules have split plausibility scores on one or more items. Nine of them have two or fewer split plausibility scores, indicating that little of the variance is due to subjects. Knowing the instantiation of these rules *would* improve our predictions of their plausibility over the predictions we can make from structure and rule knowledge. The remaining 19 rules all have between 3 and 6 split plausibility scores, which suggests that more of their variance is due to subject differences.

6.5 Conclusion

This paper suggests that we can automatically derive plausible inference rules from the relations in knowledge bases and predict judgments of plausibility for the conclusions of these rules. Two structural factors (transitivity or GPI) correctly predict plausibility 77% and 68% of the time. No knowledge is required to apply these criteria. Greater accuracy requires more knowledge, particularly knowledge about the specific rules and the concepts that instantiate them; but because we ›uld not accurately estimate the contribution of individual differences among our subjects to T, we do not know the limit on the accuracy of our predictions.

Our experiments relied on the GRANT KB, which was built for a different purpose. Although our results are limited to this knowledge base, we believe they are more general, because the surface relations in the GRANT KB are common, and because h and t are general semantic components, and because transitivity and GPI are common structural characteristics. But further work is required to *prove* the generality of our results.

Our goal was to develop methods to support graceful degradation and knowledge engineering. Clearly, these purposes are not met if plausible inference rules require masses of knowledge to judge their conclusions. We are very encouraged by the relatively high accuracy of criteria that require no knowledge, and by the fact that our accuracy is higher for plausible rules than for implausible ones.

Part V

Methodology

Chapter 7

Toward AI Research Methodology: Three Case Studies in Evaluation

7.1 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 7.3.2 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. 7.4.1). 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 7.2 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. 7.4), 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 [Buchanan, 1987], [Newell and Simon, 1976], [Langley, 1987]

[1] For example, Cognitive Science and related areas have been accused of overemphasising rigorous, reductionist methodology. Newell expresses this view in a paper called "You can't play 20 questions with nature and win" [Newell, 1973]; 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* [Neisser, 1976].

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 [Geissman and Schultz, 1988], [Gaschnig *et al.*, 1983], [Rothenberg *et al.*, 1987] 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.

7.2 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. 7.1 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 2, Fig. 7.1), we often find that the problem is ambiguous, overambitious, or underspecified and must be refined (stage 1, Fig. 7.1).

The model of empirical AI research in Figure 7.1 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 7.1–7.5 of section 7.5.

[2] Buchanan [Buchanan, 1987] describes a similar model.

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" [Lenat, 1976]. The process is iterative and directed by evaluations (as are all other stages in Fig. 7.1): 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 7.1 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 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^* [Barr and Feigenbaum, 1981] or candidate elimination [Mitchell, 1977], or Waltz filtering [Waltz, 1975]. 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 [Erman *et al.*, 1980]. 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.

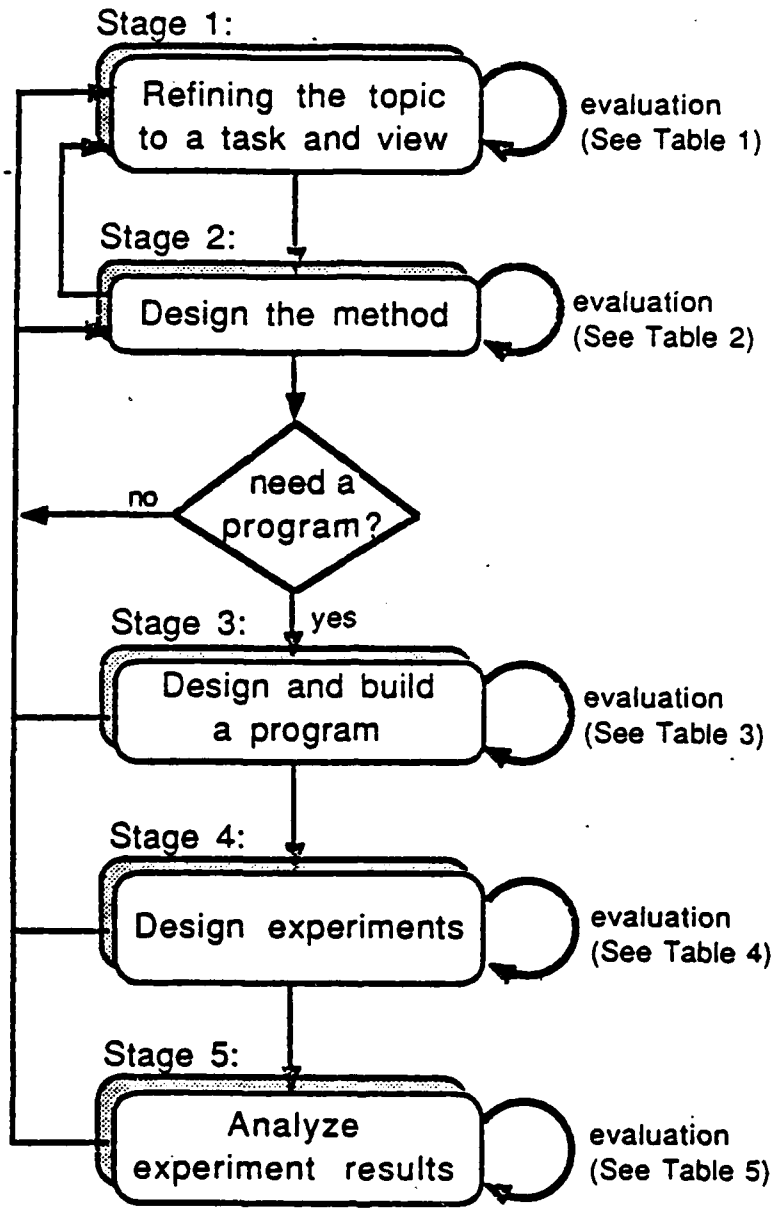


Figure 7.1: Cycle of Empirical AI Research

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 7.2 in the Appendix presents criteria that assess how well you understand your method—its strengths 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 7.1. 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 7.3 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 7.4 in the Appendix). In practice, stages 3 and 4 are interleaved: one doesn't implement programs before thinking about experiments. Section 7.3.1 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 7.5 in the Appendix). So the purposes of evaluation at this stage are to convince the research community of the viability, performance and scope of one's methods, and to suggest further research.

In sum, evaluations at each stage in Figure 7.1 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.

7.3 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 [Cohen and Lieberman, 1983]. 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 7.2. We describe how the results of analyzing Dominic, a mechanical engineering design system, led to more powerful versions of the system.

7.3.1 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 [Cohen and Lieberman, 1983]. 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, and 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 [Clancey, 1984b], 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 [Hillier and Lieberman, 1980]. 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. 7.3.2 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. 7.3.2 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 [Shortliffe, 1976]. The non-expert 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 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 de-

[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.

signed 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.

7.3.2 GRANT

GRANT finds sources of research funding given research proposals [Cohen *et al.*, 1985]. The principal difference between GRANT and most other Information Retrieval (IR) programs is that its retrieval algorithm 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 leuco-

cyte), 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. 7.3.1). 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 [Rijsbergen, 1979]. 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 [Cohen and Kjeldsen, 1987] 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 [Henrion and Cooley, 1987]. 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

[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 worthwhile. GRANT, in the early days, would find about 15 agencies of which at most 5 were *not* worthwhile.

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.

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.

7.3.3 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* [Dixon *et al.*, 1984].[5] In iterative redesign, a rough initial design is gradually improved by making small changes to the design and evaluating the effect of those changes. The method based on this view involves 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 those changes that are predicted to improve performance. This iterative

[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.

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, 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 [Howe *et al.*, 1986]. 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 7.2. Thus, the next phase of the project, given the working program, was to design, run, and analyze experiments on 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.

In Domineering, we adopted the view that if Dominic could design artifacts that performed to problem specifications, then it should be able to design itself to perform well in specific domains. Domineering was built

to learn the best configuration of Dominic-I for particular design domains [Howe, 1986]. In effect, Domineering is Dominic-I applied to itself. Dominic is used to design the best configuration of Dominic. The method required is nearly identical to Dominic-I except that the alternate control strategies mentioned above provided the design variables, the performance criteria provided the goals, and Domineering could call Dominic-I to implement and evaluate changes. 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 performed better, 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. 7.1), we had completed the last stage and were poised to begin the cycle anew. For Orelup and Dixon, who continued the project, Dominic-II [Orelup, 1987] 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. So Orelup and Dixon 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.

7.4 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, fails 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. Fortuitously, the excerpted research draws on several fields, including physiological, abnormal, and developmental psychology.[6] Hereafter, we refer to the excerpt as the autism article.

> 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, $F(4,29) = 3.57$, $p < .02$. [Dawson *et al.*, 1986]

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

[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.

(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 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, rep-

resentations, ..., 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., [Boden, 1979] 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* [Mitchell *et al.*, 1986] or *deductive* [Kemeny, 1959]. 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.

7.4.1 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. 7.3.3) 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 consentual, 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 7.3.1 and 7.3.2, 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" [Newell, 1975]) is informative in systems that can solve problems without them. For example, one configuration 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.

[7] Shortliffe ran a panel of experts and novices in his studies of MYCIN [Shortliffe, 1976].

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.

Which of these, if any, are appropriate for any given project will depend on the project and its research goals. Inductive studies may be too difficult for systems that require a tremendous amount of domain knowledge or knowledge that is difficult to obtain. Moreover, because these schemas were culled from experiments on self-contained expert systems, issues common to other types of knowledge-based systems have not been addressed. For example, systems that interact with other systems (e.g., humans in mixed-initiative dialog and programs in distributed processing) add another level of complexity to evaluating and understanding their performance. Clearly, design schemas must be developed to respond to the special needs of specific areas of AI.

7.4.2 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 7.2. 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 [Buchanan and Shortliffe, 1984]. 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

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

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, 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.

[9]Lately, the machine learning community has mentioned evaluation criteria explicitly in Calls for Papers, and Langley has advocated evaluation in his editorials [Langley, 1987].

7.5 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 7.1: 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 7.2: 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 7.3: 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 7.4: 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 7.5: Criteria for evaluating what the experiments told us

Chapter 8

Why Knowledge Systems Research Is In Trouble, And What We Can Do About It

8.1 Introduction

I was recently reminded of Allen Newell's paper *You Can't Play 20 Questions With Nature and Win.* Newell wrote it in his role as discussant at a Carnegie Symposium, and addressed it to the cognitive psychology community. The following paragraph appears early in his comments:

> I was going to draw a line on the blackboard and, picking one of the speakers of the day at random, note on the line when he got his PhD and the current time (in mid-career). Then, taking his total production of papers like those in the present symposium, I was going to compute a rate of productivity of such excellent work. Moving, finally, to the date of my chosen target's retirement, I was going to compute the total future addition of such papers to the (putative) end of this man's scientific career. Then I was going to pose, in my role as a discussant, a question: Suppose you had all these additional papers ... *where will psychology then be?* Will we have achieved a science of man adequate in power and commensurate with his complexity. And if so, how will this have happened via these papers I have just granted you? Or will we be asking for yet another quota of papers in the next dollop of time.

– Allen Newell, You Can't Play 20 Questions With Nature and Win. In W.G. Chase (Ed.) *Visual Information Processing.*

When I read Newell's paper for the first time, I was an interdisciplinary graduate student in psychology and AI. The attitude toward psychology in the Stanford Heuristic Programming Project convinced me that a truly interdisciplinary thesis wasn't possible, and Newell's paper was an important reason that I chose to work in AI and not psychology. I have always been strongly influenced by the questions Newell asks in the quote above, and lately I have started to think they apply to AI as well; and if not to AI in general, certainly to expert systems. Add up all the papers on expert systems in IJCAI, AAAI, and AI Magazine over the last five years, and what have we got? What can we expect in 1995? When Newell asks whether psychology will ever add up to a "science of man," he assumes a common purpose among psychologists. Whether or not this is realistic in psychology, it certainly doesn't characterize expert systems research. I can't see any common scientific purpose there, which is one reason I think expert systems research isn't adding up to a science of anything. Newell asks whether individual papers, each excellent and methologically sound, add up to a science. I don't think we have more than a handful of excellent and methodologically sound papers, period. These two issues—scientific purpose and methodology—are at the heart of my dissatisfaction with expert systems.

The following two subsections are about these issues. Let me preface them by characterizing my biases. First, when I talk about research in expert systems or knowledge systems, I am referring to a range of component technologies, including knowledge representation, learning, ontology, case-based reasoning, generic tasks, and control. It is easy to criticize hack work in expert systems, and my comments can certainly be interpreted this way (i.e., as criticisms of the other guys), but I direct them primarily to my colleagues in the research community. Second, I view expert systems as a vehicle for Artificial Intelligence research, not as an engineering subdiscipline. Irrespective of the value of expert systems in the marketplace, they are worthless to me unless they tell me something about intelligence. Third, I think of intelligence primarily in terms of behavior—what an agent thinks or does—and secondarily in terms of structures—what the agent knows, or how it is represented. With these biases, which I think are pretty common, let me now say why expert systems research has become intellectually unsatisfying.

8.1.1 Learning about Intelligence

I am not learning much about intelligence, artificial or otherwise, from the knowledge systems literature. One reason is that we have come to equate intelligence with knowledge, which is static, instead of with dynamic behavior. This is seen most clearly in the disregard for control in the knowledge systems community, although the consequences aren't felt until someone tries to acquire and reason with knowledge about how experts (and expert systems) should behave. For example, when Clancey tried to represent diagnostic strategies for the purpose of tutoring, he discovered (as we did later in our MUM project [Cohen *et al.*, 1987a]) that the only ways to specify the behavior of knowledge systems were brutish chaining, meta-rules, bizarre hacks, or lisp programming [Clancey, 1986]. If you are looking for tools to describe behavior, the knowledge system literature offers little besides reassurance that simple control strategies are sufficient. Sufficient for what? Certainly not for modelling the complex behavior that Barbara Hayes-Roth observed in human errand planning [Hayes-Roth and Hayes-Roth, 1979]; or that Ed Durfee requires for simple cooperative, distributed problem solving [Durfee and Lesser, 1987]; or for the simple meta-planning we require to prune the search space in mechanical design [Orelup *et al.*, 1988]; or for real-time planning against simulated forest fires (see below).

Research on reasoning under uncertainty is another, personally frustrating, manifestation of the emphasis on knowledge and structure over behavior and dynamics. It offers more and more refined ways to calculate degrees of belief, but relatively little work on how problem solvers plan, decide, and act in uncertain environments [Cohen and Day, 1988], [Cohen, 1987a]. Once again, intelligence is equated with what you know (more precisely, with maintaining degrees of belief in propositions), not with how you behave.

The uncertainty community evidently feels it is subcontracting to the knowledge systems community. They are the folks who provide the calculus. Judging by the preambles of papers I have recently reviewed, a lot of people think this way. "Knowledge systems need my innovation in ...." You can fill in the dots with uncertainty calculus, explanation mechanism, learning mechanism, reason maintenance system, and so on. The knowledge systems literature, and AI in general, is busily producing component technologies. But it is difficult to learn much about intelligence from a bunch of component technologies. They don't add up to a theory of anything, any more than the unremitting tide of psychology papers add up to "a science of man." (There are also mundane, pragmatic reasons to worry about the component technology approach to knowledge systems, as I'll discuss later.)

Another reason that knowledge systems tell me little about intelligence is that they have trivial environments. In the days before knowledge systems, we believed that intelligent behavior emerges from the interaction between the structure of an agent and its environment. That's what Simon's Ant was all about. This view has a simple but important implication: Any study of intelligence that ignores the environment is underconstrained. This gives rise to the queasy feeling that I expect you have, from time to time, when you review papers: Why did the author do things this apparently arbitrary way? Here's an example, drawn almost at random, from a batch of IJCAI papers I just reviewed:

> The strategic planner is goal-oriented, the tactical planner is resource oriented, and the reflexive planner is event (signal-) oriented. A mobile agent must incorporate all three types of planners. ...There are two principal techniques to implement [the tactical] planner: opportunistic and least-commitment planning. In our approach, we prefer the opportunistic planning technique.

Note the imperative—a mobile agent *must* incorporate all three types of planners—and the arbitrary preference for opportunistic planning. What aspects of the task environment engender the imperative and justify the preference? If you design a knowledge system without first considering how its environment constrains the design, your design will be arbitrary.

Unfortunately, most knowledge systems are intentionally isolated from their environments, and thus have underconstrained and arbitrary designs. Environments are typically a couple of dozen categorical propositions: the patient has a temperature, the organism is aerobic, a blip was reported at latitude x and longitude y. Few knowledge systems are designed for dynamic, uncertain, or multi-actor environments; and fewer still can manage real-time constraints. Rick Hayes-Roth went so far as to advise British companies, in a Pergamon report, to "Seek problems that experts can solve via telephone communication." [Hayes-Roth, 1984]. That is, build knowledge systems that don't need to know much about the external environment. This attitude ruined planning research, where for years the best systems were built for environments that simply don't exist: environments in which the planner is the only agent, in which actions are instantaneous, and their effects persist indefinitely; environments in which the state of the world and the effects of all actions are known or accurately predictable. NOAH's environment was quasi-static, and contained three unambiguously-labelled blocks and a table top. Why did we regard NOAH as the apex of planning research for so many years?

Many of these arguments are summarized in Figure 3.1, which is a crude history of how AI has characterized intelligence. Initially, intelligence was viewed as the behavior that emerged from the interaction between autonomous agents and their environments. But knowledge systems later became isolated from their environments, behavior was de-emphasized, and research on complete, autonomous agents was replaced by work on component technologies. Answers mattered; the process of deriving them did not. The majority of tasks were "one-shot," meaning that we would solve the problem now, once and for all, and not monitor or revise the solution in future. One-shot problems denied us the opportunity to study behavior in ongoing environments; for example, instead of building expert systems to "wait and see" how a patient's condition develops, we built systems to give the best possible recommendation *now*, even if the data were poor.[1] Again, this is seen clearly in the uncertainty literature, where virtually all the research concerns what to do with the evidence you already have, and none concerns how to get evidence, corroborate it, hedge against future outcomes, or other strategies for coping with ongoing, uncertain environments.

More recently, there has been a renaissance of old ideas about intelligence. A few projects are beginning to acknowledge the role of the environment. The planning literature has been reinvigorated by ideas about situated action [Georgeff and Lansky, 1987]. Major DARPA-sponsored efforts, such as Pilot's Associate and the Autonomous Land Vehicle, are forcing us to contend with dynamism, real time, and multiple actors. But in the following section, I will ask whether we have the methodology to capitalize on this positive change in emphasis, or whether we will end up doing the same kind of inconclusive, "look ma, no hands" research in yet another set of task domains.

8.1.2 Methodology

One methodological problem for knowledge systems research is that the ratio of science to engineering is too low. Back in the old days, it took many years to build an expert system, but one at least had the chance of discovering something. Today, knowledge systems are bigger, and still take years to build, but they are rarely built to discover anything. One has to work very hard to get a system that does ...pretty much what one expects

[1] You might object that sometimes we can't afford to wait and see. I agree that we need to build systems that reason about what they afford (e.g., following Lesser's idea of approximate processing [Lesser *et al.*, 1988]). This is precisely the kind of reasoning that has been absent from knowledge systems until recently.

it to do. One can't dismiss this as applying only to hack applications; with few exceptions, none of us are discovering enough to warrant the engineering effort of our projects.

Knowledge systems are built as demonstrations, not as experiments. Researchers rarely say what they intend to learn by doing their work, or what they actually learned by doing it. More often, proposals and papers assert that "We need X, and here's how we expect to provide X, and (later), here's a demonstration of X." Lenat and Feigenbaum put it this way:

> If one builds programs that cannot possibly surprise him/her, then one is using the computer either (a) as an engineering workhorse, or (b) as a fancy sort of word processor (to help articulate one's hypothesis), or, at worst, (c) as a (self-) deceptive device masquerading as an experiment. [Lenat and Feigenbaum, 1987]

Their alternative, the empirical inquiry hypothesis, says what we should be doing although not how to do it:

> The most profitable way to investigate AI is to embody our hypotheses in programs, and gather data by running the programs. ...Progress depends on the experiments being able to falsify our hypotheses; i.e., these programs must be capable of behavior not expected by the experimenter.

What hypotheses? The general form of a hypothesis in knowledge systems research is "X is sufficient to produce Y"; for example, a rule-based representation of expert knowledge and a backward chaining interpreter are sufficient to produce therapy recommendations at an expert level. Given the emphasis on component technology, mentioned earlier, most hypotheses are more specific; for example, the Dempster-Shafer method is sufficient to combine evidence in MYCIN. But unlike hypothetico-deductive science, we never show the necessity of one hypothesis by *rejecting* a mutually exclusive one. Our principle mode is to accept the null hypothesis, to accrue demonstrations of sufficiency.

It doesn't have to be this way, and the few counterexamples suggest our science would be more productive if we tried more often to show that mechanisms *don't* work. I think the best results of Lenat's work with EURISKO and AM emerged from his failed attempts to apply AM'S techniques to heuristics themselves. The failure, and the ensuing enquiry, led to this remarkable conclusion:

> It was only because of the intimate relationship between Lisp and Mathematics that the mutation operators (loop unwiding, recursion elimination, composition, argument elimination, function substitution, etc.) turned out to yield a high "hit rate" of viable, useful new math concepts when applied to previously-known, useful math concepts—concepts represented as Lisp functions. But no such deep relationship existed between Lisp and Heuretics, and when the basic automatic programming (mutations) operators were applied to viable, useful heuristics, they almost always produced useless (often worse than useless) new heuristic rules. [Lenat and Brown, 1984]

This is one of the few strong results of knowledge systems research, one of the few papers I would cite if asked what we have learned by building all these systems.

There are really two, interrelated methodological issues here. One is how we select research problems, the other is what we do with them. We are very good at selecting new research problems because we are very poor at studying them. Instead of trying to find out why something works or doesn't work, as Lenat and Brown did, we are content to show merely that something works. Once we have demonstrated sufficiency, we move on to another problem. I call this the strip mining heuristic: Once you have grabbed the gold near the surface, move on. The gold nearest the surface is by convention demonstrations of sufficiency. Knowledge systems research trashes the space of questions about intelligence in much the way that slash-and-burn cultures trash the rain forest. Both make very inefficient use of resources and impress upon me a horror of waste. Even when the resources are used well and we get all we can out of each project, as in Buchanan and Shortliffe's superb collection of papers on MYCIN [Buchanan and Shortliffe, 1984], most of our work is still demonstrations of sufficiency.

With Adele Howe, I have started to outline methods for getting results out of knowledge systems [Cohen and Howe, 1988b], [Cohen and Howe, 1988c], and I know that other areas of AI are engaged in similar efforts [Langley, 1987]. All this work is pretty preliminary, and it needs to be done whether one works with knowledge systems or in some other area of AI. I still spend a lot of time wondering how to do research instead of doing it. But this is preferable to messing around with expert systems as we have in the past.

8.2 What Now?

In the previous section I raised two issues, scientific purpose and methodology, that cause me (and I believe others) to be dissatisfied with knowledge systems research. In this section I describe a problem that, I believe, focusses my research on the right scientific issues, and the methodology that I think is appropriate to study it.

We have built a large simulation of forest fires and the equipment commonly used to put them out. We are building a planner called PHOENIX that operates in this dynamic, real-time world. The planner's goal is to manage the fire—limit the loss of human life, limit the damage to forest and buildings, and limit the monetary costs of achieving these goals. Our simulation consists of a large geographical area ("Explorer National Park") in which there is a considerable variety of topography and ground cover, as well as roads, lakes, and streams. These features affect how forest fires burn. Equally important features are wind speed and direction, both of which can change unpredictably; and the moisture content of the ground cover, which varies in time and geographically. To fight the fire, the simulation provides bulldozers, crews, transport vehicles, planes and helicopters. These cut fire line, move firefighters, spray water, or dump retardent.

Originally, these fire-fighting agents were directed by a human player in what was essentially a complex, real-time video game. We gained considerable insight into (and respect for) the dynamics of this mini-world by playing against the simulation—often losing many lives and considerable real estate to a seemingly slow and containable fire. It is difficult for a planner, human or AI program, to do very well at the game (i.e., put out the fire with reasonable costs, no loss of life, etc.) because:

- The simulation is real-time with respect to the fire. While fire-fighting agents move, cut line and drop retardent, the fire keeps burning. Any time the planner devotes to deliberation is claimed as real estate by the fire.

- The player's knowledge of the fire is limited to what the agents in the field can "see." Crews and bulldozers can see only short distances; watchtowers and aircraft can see further. The planner rarely, if ever, has complete knowledge of the extent or location of the fire. This is evident in Figure 8.2, which shows in the right panel the world as it "really is," and in the left panel the world as seen by a watchtower—the spindly icon a little north of the center of the panel. (The figure

also shows roads, houses, and a lake in the southeast. This is a small fraction of Explorer National Park. Normally, the display is in color.)

- The behavior of the fire cannot be accurately predicted because some factors that affect it, terrain, ground cover and the moisture content of the ground cover, are known only approximately. Moreover, wind speed and direction can change unpredictably.

- The behavior of the fire-fighting agents cannot be accurately predicted. In particular, the time required to move to a location or perform some task depends on terrain and ground cover. Fire-fighting agents also have varying degrees of autonomy, so the central planner cannot always be sure of their location.

These are some of the specifics of the fire environment, and the difficult technical problems they raise for the PHOENIX planner. More generally, they exemplify the kind of environment we should be studying in knowledge systems research if we want to learn about intelligent behavior. Two salient characteristics of the fire environment are uncertainty and real-time dynamics, described above. Others are that the environment is *ongoing* (as opposed to one-shot), so the emphasis is on controlling a process through one's behavior, as opposed to solving a problem through inference. The environment supports *multiple agencies*, such as wind, rain, and the fire itself; and it supports multiple actors which need to be coordinated to get a global view of the situation and to control it. The environment also has several measures of success (or failure). Most importantly, it doesn't have just one "right answer," but requires a planner to evaluate tradeoffs between plans, even during execution.

Given this focus, how should we proceed? We have three goals, related to the problems we discussed earlier:

- Work in complex environments, in which behavior matters.

- Justify design decisions by reference to aspects of the environment, instead of accepting the first sufficient design.

- Design and build complete, autonomous agents, and de-emphasize component technologies.

I am hedging my bets with respect to these goals by working simultaneously within two methodological frameworks. The first, which guides the development of PHOENIX, is a *top down* design effort in which we identify

the abilities that we believe a planner will need to excel in the fire domain. We designed the domain itself to ensure that providing these abilities would solve open technical problems in AI (specifically, problems in real-time planning and distributed AI). As predicted by the empirical inquiry hypothesis, we are discovering unexpected behaviors. One surprise is that purely reactive behavior is sufficient to put out some fires. But to follow through on the goals, above, we need to explain this result by reference to aspects of the environment. For example, one reason that reactive planning works is that the fires are typically convex, so it is rare for reactive bulldozers to get trapped in "pockets" of fire (but see Figure 8.3) for a counterexample.) Note also that we can't explain getting trapped (or avoiding it) by reference to any single component of the bulldozer's architecture. Getting trapped is a function of the bulldozer's radius of view, the frequency with which it updates its view, and its set of reactions. We can't explain performance in terms of a single component, nor can we improve performance by developing component technologies in isolation.

I call PHOENIX a top-down design effort because it is driven by a longish list of design goals. Eventually, PHOENIX will handle multiple fires, and coordinate multiple fire-fighting objects. It will monitor the progress of plans in real-time, and modify plans to give the best performance for the available resources. Where do these goals come from? One source is the judgment that the PHOENIX planner will need these skills to put out fires; the other is the recognition that these are open technical problems in AI. Now, ordinarily, a researcher is congratulated for finding a task, like fire-fighting, that requires methods which AI hasn't yet developed. Fire-fighting is a good task because, apparently, we will need to advance the state of the art to do it right. But in the context of my previous comments, I think we should be a bit suspicious. Are these design goals really mandated by the environment? I believe that these skills will enable a planner to put out fires (i.e., will enable a demonstration of sufficiency), but are they *necessary*? Do we need to modify plans during execution, or is this just a bit of technical showmanship? The trouble with top-down research is that this question is often very difficult to answer. For this reason, I have recently started another project with the same methodological goals—to work in complex environments, eschew component technology, and justify design decisions in terms of structure and dynamics of the environment—but with a different methodology.

We can approach intelligence from the bottom up by starting with simple structures, extending them only to provide adaptability in environments, but

making the environments increasingly complex, and the required behaviors increasingly sophisticated. Bottom-up work is rare in AI, but is typified by Brooks' approach to robotics [Brooks, 1986] and Braitenberg's Vehicles [Braitenberg, 1984]. Bottom-up research is applied in the sense that all our technology must be immediately useful, that is, must have "adaptive significance" to the automata for which it is developed. We don't introduce technology for its own sake, but only to make our automata more capable in increasingly complex environments. Progress is driven by primarily by environments.

Perhaps the most important aspect of bottom-up research is that it uncovers *unintended, emergent* behavior. By emergent I mean behavior that is due to interactions between the agent and its environment, or within the agent. Our recent work suggests that unintended, emergent behavior is common in agents that interact with even simple environments over time. Moreover, this behavior has assymetric consequences for the bottom-up and top-down research strategies. For bottom-up research, every emergent behavior is an opportunity to expand the repertoire of behaviors; that is, many emergent behaviors are serendipitous. For top-down research, unintended, emergent behavior is rarely serendipitous. It usually messes up the design. This suggests that if we are trying to design agents to interact with ongoing, dynamic environments, it may be more efficient to design them bottom-up than top-down. For example, imagine I wanted an automaton to learn how to find its way downhill from any point in a landscape to the lowest accessible plateau, and to stay on the plateau thereafter. A top-down approach might involve three components—one to get the automaton onto a plateau (presumably using what it had previously learned), one to learn from the current problem, and one more to keep the automaton on the plateau once it gets there. In fact, a recent bottom-up approach to this problem found that the third component is unnecessary: in learning how to go downhill, the automaton learns not to go uphill, and so by the time it reaches the plateau it already knows enough to avoid moving off the plateau.

The bottom-up project, which we call PM, is in its early stages. Eventually, we hope to have autonomous agents capable of putting out fires in Explorer National Park, but the methodology calls for approaching this goal incrementally, so now we are working with simple automata in simple environments. For example, Figure 8.4 shows a "beach" over which a version of Simon's ant perambulates.

Recall that one of my three methodological goals is to build complete automata. We don't really know the minimum set of skills an automaton

needs, but I settled on these four as a first cut:

1. Automata should perceive their environments. Roughly, this means at least that they construct internal representations of some or all of their environments.

2. Automata have internal state. In some cases, internal state will be no more than the internal representation of the environment. In others, it will include "forces" like "hunger." In more sophisticated automata, internal state might result from processing information beyond perceptual processing. Internal state (including perceptions) determines how automata behave.

3. Automata act. This may be definitional, since the state of an object that doesn't act is determined exclusively by its environment, so that object isn't autonomous.

4. Automata learn.

Since the inception of the project, we have designed a dozen automata for the environment in Figure 8.4, and for related environments that present more difficult learning and performance problems. We have observed automata getting stuck in corners, getting trapped on plateaus, exhibiting "superstitious" behavior on downhill trajectories, decreasing their rate of learning, and cycling and other repetitive behaviors. These were all unintended behaviors. There was no way to predict them by looking at the automaton's code, nor were they intended by the programmer. They are emergent behaviors in this sense: None can be explained by a single aspect of the automaton's design. Take superstitious behavior as an example. When an automaton moves downhill, it remembers the context in which it started the move, and the move itself, and increases the score of that move in that context. Furthermore, it always selects the move with the highest score. This means that the first positively-scored move will have a higher score than any other, and will always be selected in that context. Moreover, because of the way the environment is constructed, and because of the way contexts are constructed to access moves in memory, a move that led downhill in a context will typically lead downhill the next time the context is encountered, so the move will typically have its score increased each time it is repeated. Note that to explain a single observed behavior, I have had to discuss the agent's learning mechanism, the structure of the environment, how memories are accessed, and how moves are selected. For brevity, I left

out the influence of the perceptual system, but it too plays a role in the emergent behavior.

These behaviors aren't necessarily desirable, but after years of AI programs that do *exactly* what's expected, any surprises are refreshing. More to the point, the desirability of emergent behaviors depends on the environment; superstitious behavior on downhill trajectories is only a problem if the environment demands that automata find the *fastest possible* path down a hill.

For each automaton, in each environment, we ask some or all of these questions:

1. What is the minimum structure necessary to achieve a level of adaptation for an automaton in an environment?

2. How robust is the automaton to ranges of environmental conditions?

3. Could the amount of learning necessary to achieve adaptation be reduced by making another evolutionary step? What other aspects of the interaction between environment and automaton seem to require a more sophisticated automaton?

4. As much as possible (given emergent behavior, interactions, and nondeterminism) explain *why* the agent behaves as it does in particular environments. This is especially important for emergent (unexpected) behaviors. Why does the ant describe a sawtooth? Why does it get stuck in corners? Are these behaviors adaptive, given one's definition of adaptive?

5. Are there parallels between the design of one's automaton and biological systems? There needn't be, and we'd be fools to reject designs because they couldn't occur in organic systems, but if the parallels are there, I'd like to know about them.

To date, the PM project has not achieved any major results, but we have discovered some minor ones. For example, an automaton with a relatively poor ability to discriminate contexts can actually outperform automata with greater acuity. This is because an inability to distinguish contexts is de facto generalization over contexts, so automata that can't distinguish specific situations in effect learn classes. A simple example is the "beach" in Figure 8.4. All our automata learn which of eight neighboring cells to visit from any given cell, and all construct a context for a move from the altitude values of the eight neighboring cells. But the original automata discriminated

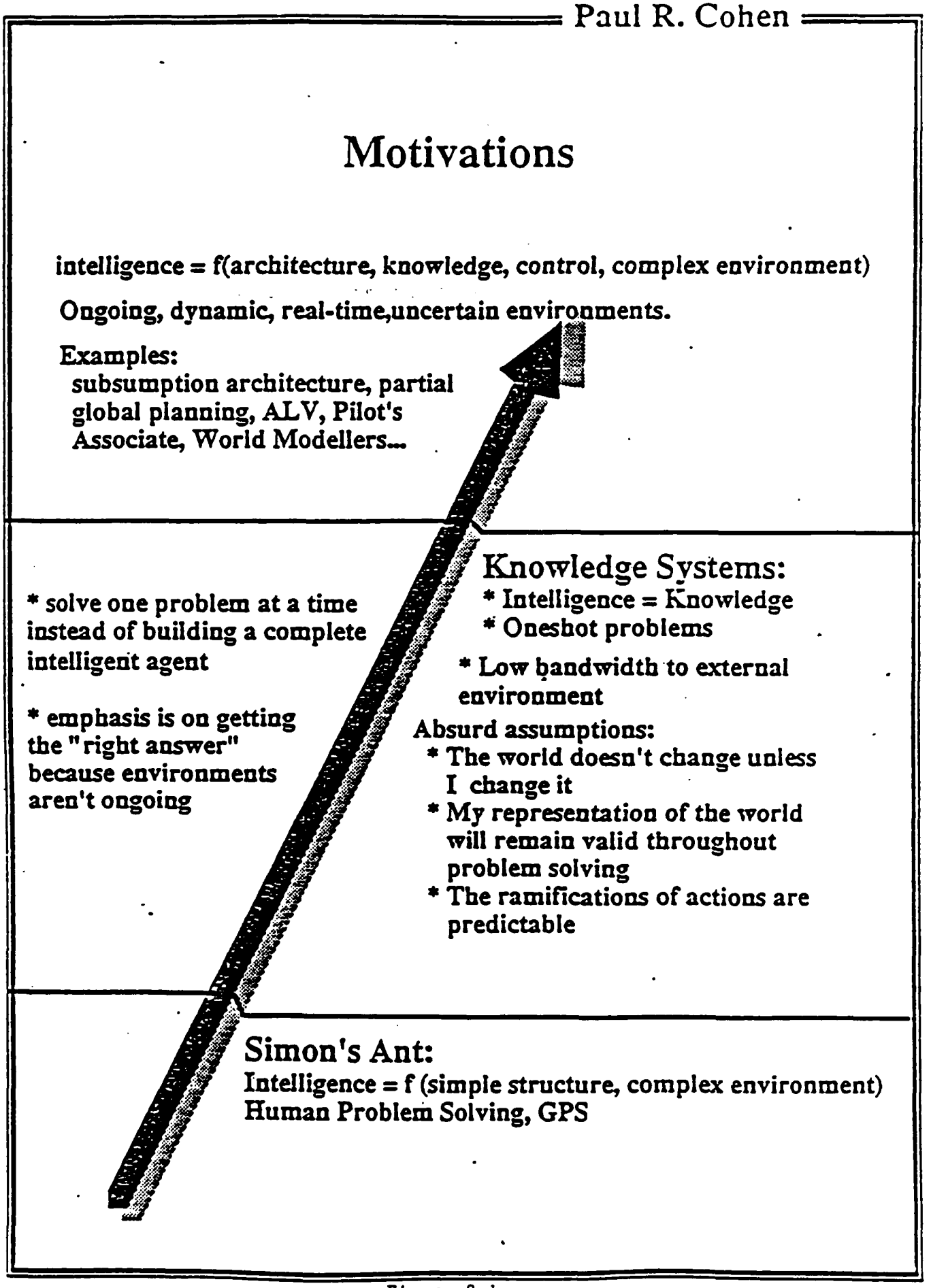

Motivations
intelligence = f(architecture, knowledge, control, complex environment)
Ongoing, dynamic, real-time,uncertain environments.
Examples:
subsumption architecture, partial global planning, ALV, Pilot's Associate, World Modellers...
Knowledge Systems:
* Intelligence = Knowledge
* Oneshot problems
* Low bandwidth to external environment
Absurd assumptions:
* The world doesn't change unless I change it
* My representation of the world will remain valid throughout problem solving
* The ramifications of actions are predictable
* solve one problem at a time instead of building a complete intelligent agent
* emphasis is on getting the "right answer" because environments aren't ongoing
Simon's Ant:
Intelligence = f (simple structure, complex environment)
Human Problem Solving, GPS

Figure 8.1

contexts by the actual altitude values, while later automata simply asked whether neighboring cells were "up" or "down" from the current cell. The latter case is de facto generalization over contexts. Moreover, it is a good generalization, because the automata are punished or rewarded not for moving to particular altitudes, but for moving up or down. Now, in the long run, the automaton that can discriminate more contexts will outperform the one that discriminates fewer; but it takes a lot longer to learn all the detailed contexts. You can have performance at one level quickly, or you can wait longer and get higher performance. The choice is determined by the environment, as all design decisions should be.

8.3 Conclusion

I started this paper with Newell's provocative questions to the psychology community, and I want to end it by reviewing his recommendations. I found it remarkable that his questions were so pertinent to AI, and equally remarkable that his advice to the psychology community is so pertinent to us. Here is what Newell recommends:

> The first recommendation is to construct complete processing models, instead of partial ones as we do now. . . . The second . . . is to accept a single complex task and do all of it. [Newell, 1973]

Newell's paper was written 15 years ago, when psychology seemed stuck and AI seemed to offer an alternative. Today, knowledge systems research seems stuck because we *haven't* built complete processing models, but have focussed on component technologies; and because we haven't accepted complex tasks, but have stayed within trivial environments. If we are to fare better than cognitive psychology, it's time to follow Newell's advice.

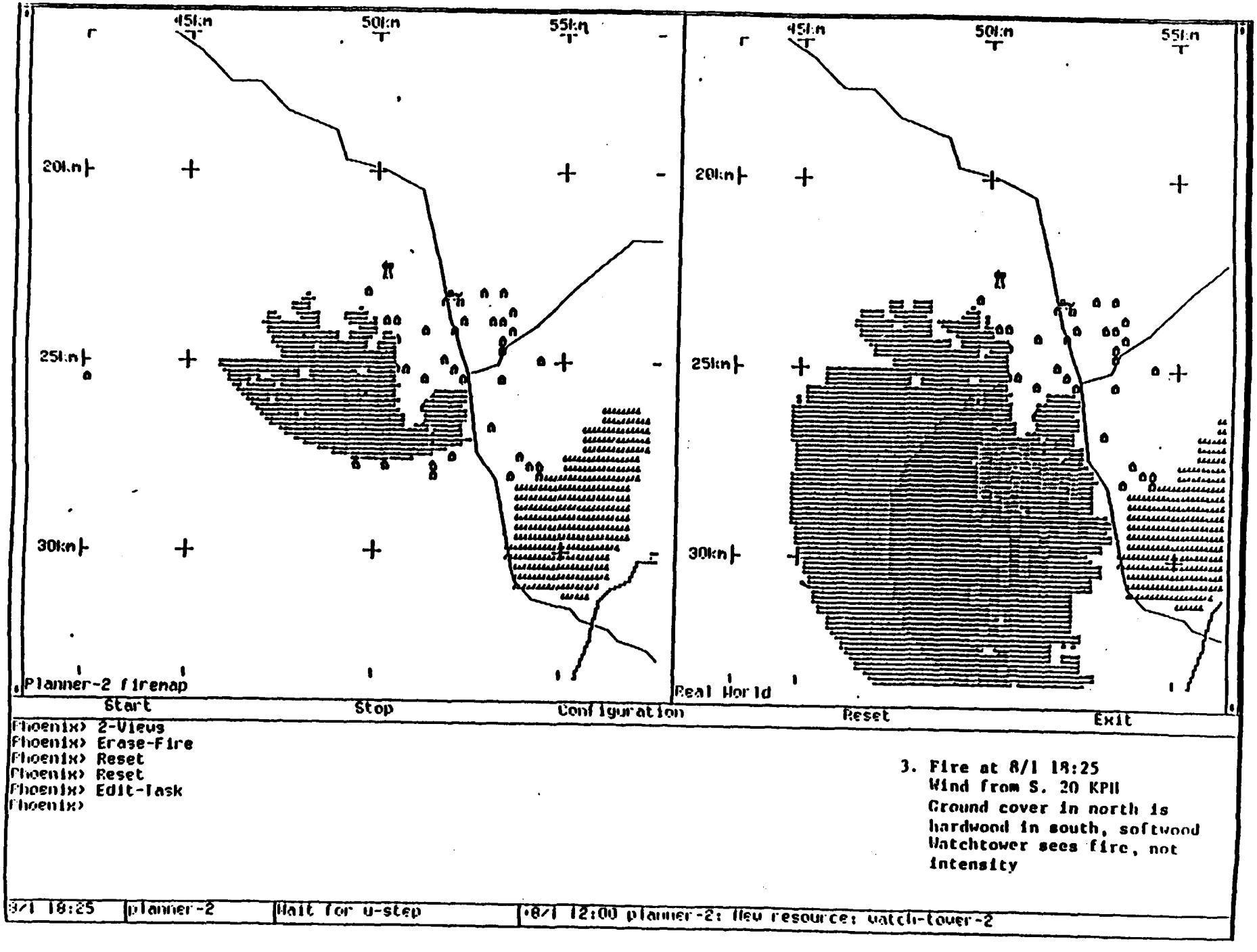

45km
50km
55km
20km
25km
30km
Planner-2 firemap
Real World
Start
Stop
Configuration
Reset
Exit
Phoenix> 2-Views
Phoenix> Erase-Fire
Phoenix> Reset
Phoenix> Reset
Phoenix> Edit-Task
Phoenix>
3. Fire at 8/1 18:25
Wind from S. 20 KPH
Ground cover in north is hardwood in south, softwood
Watchtower sees fire, not intensity
8/1 18:25
planner-2
Wait for u-step
8/1 12:00 planner-2: New resource: watch-tower-2

FIGURE 8.2

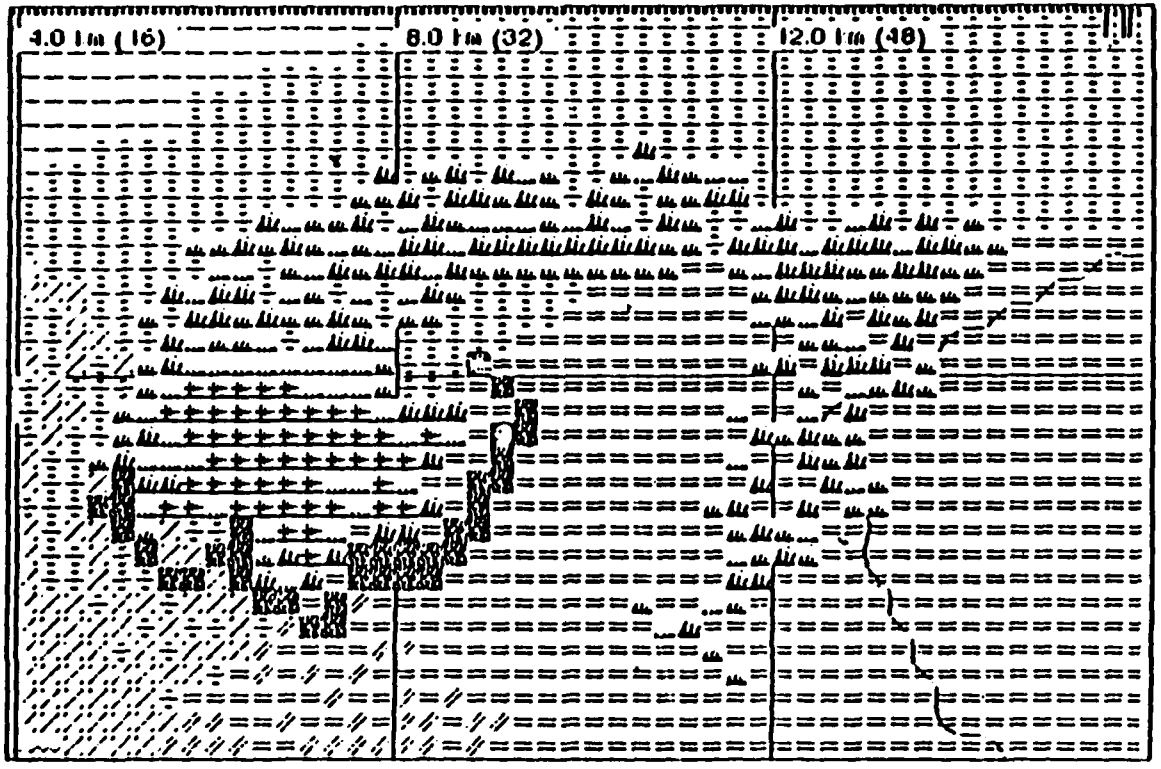
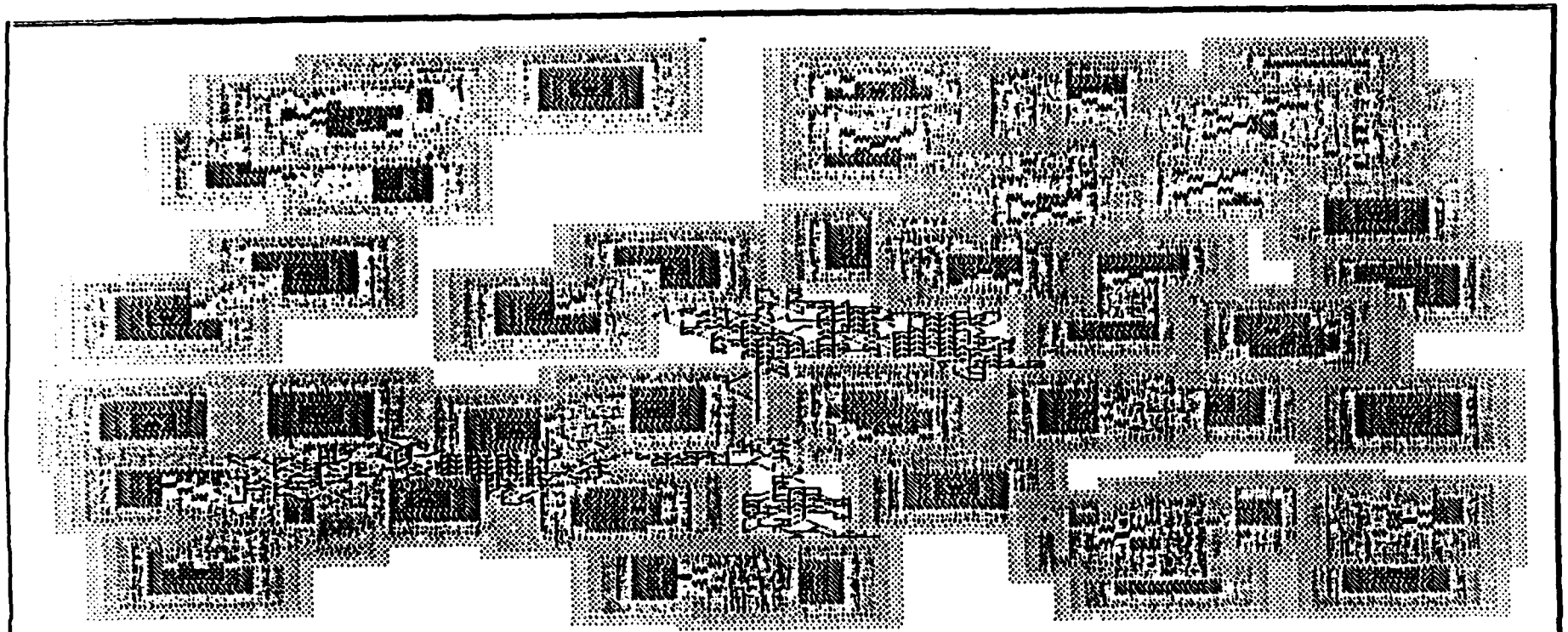


FIGURE 8.3



```
Ant Map
> (print-world-array)

nil
>
> (set-parameters)

nil
> (ant-loop)
 learning method used: colen's original method, cell value modification
The initial location is 54,51:
The initial location is 23,79:

nil
>
```

Ant Screen 2

FIGURE 8.4

Part VI

References

Appendix A

Paper References

All of these chapters were written with DARPA support. Additional support is credited below. Most of the chapters in this report have been drawn from published papers and technical reports. The references are:

Chapter 2 - [Gruber, 1989]. This work was supported in part by ONR University Research Initiative contract N00014-86-K-0764.

Chapter 3 - [Cohen *et al.*, 1988]. This work was supported in part by ONR contract AFOSR 4331690-01.

Chapter 4 - [Cohen and Day, 1988]. This work was supported in part by ONR University Research Initiative contract N00014-86-K-0764 and Naval Underwater Research Systems Center contract N66604-6288-1301

Chapter 5 - [Cohen *et al.*, 1989]. This work was supported in part by ONR University Research Initiative contract N00014-86-K-0764, and Office of Naval Research contract N00014-87-K-0238.

Chapter 6 - [Cohen and Loiselle, 1988]. This work was supported in part by ONR University Research Initiative contract N00014-86-K-0764 and by a gift from Tektronix.

Chapter 7 - [Cohen and Howe, 1988a]. This work was supported in part by ONR contract AFOSR 4331690-01 and by ONR University Research Initiative contract N00014-86-K-0764.

Bibliography

[Agre and Chapman, 1987] Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 268–272, Seattle, Washington, 1987. American Association for Artificial Intelligence.

[Allen, 1984] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154, 1984.

[Anderson, 1983] John R. Anderson. *The Architecture of Cognition.* Harvard University Press, Cambridge, MA, 1983.

[Anderson, 1986] Charles W. Anderson. Learning and problem solving with multilayer connectionist systems. Technical Report COINS 86-50, University of Massachusetts, 1986. Ph.D. Thesis.

[Ashley, 1987] Kevin D. Ashley. *Modelling Legal Argument: Reasoning with Cases and Hypotheticals.* Doctoral disssertation, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, 1987.

[Baker and Burstein, 1987] Michelle Baker and Mark H. Burstein. Implementing a model of human plausible reasoning. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 185–188, Milan, Italy, 1987.

[Bareiss *et al.*, 1987] E. R. Bareiss, B. W. Porter, and C. C. Wier. Protos: An exemplar-based learning apprentice. In *Proceedings of the Second AAAI Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff, Canada, October 1987. To appear in the *International Journal of Man-Machine Studies.*

[Bareiss, 1989] E. R. Bareiss. *Exemplar-based Knowledge Acquisition: A Unified Approach to Concept Representation, Classification, and Learning.* Academic Press, New York, 1989. Based on doctoral dissertation, Department of Computer Science, University of Texas, Austin.

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

[Barto *et al.*, 1983] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-13(5), September 1983.

[Benjamin, 1987] D. Paul Benjamin. Learning strategies by reasoning about rules. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 256–259, Milan, Italy, 1987.

[Bennett, 1985] J. S. Bennett. Roget: A knowledge-based consultant for acquiring the conceptual structure of a diagnostic system. In *Journal of Automated Reasoning*, volume 1, pages 49–74, 1985.

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

[Boose and Bradshaw, 1987] J.H. Boose and J.M. Bradshaw. Expertise transfer and complex problems: Using aquinas as a knowledge acquisition workbench for expert sytems. *International Journal of Man-Machine Studies*, 26(1):21–25, January 1987.

[Brachman and Schmolze, 1985] R.J. Brachman and J.G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2), March 1985.

[Braitenberg, 1984] Valentino Braitenberg. *Vehicles: Experiments in Synthetic Psychology.* MIT Press, 1984.

[Brooks, 1985] Rodney A. Brooks. A robust layered control system for a mobile robot. A.I. Memo 864, Massachusetts Institute of Technology, 1985.

[Brooks, 1986] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1), March 1986.

[Broverman and Croft, 1987] Carol A. Broverman and W. Bruce Croft. Reasoning about exceptions during plan execution. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 190–195, Seattle, Washington, 1987.

[Buchanan and Shortliffe, 1984] 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.

[Buchanan *et al.*, 1983] B.G. Buchanan, D.K. Barstow, R. Bechtel, J. Bennett, W. Clancy, C. Kulikowski, T. Mitchell, and D. A. Waterman. Contructing an expert system. In F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, editors, *Building Expert Systems.* Addison-Wesley, Reading, MA, 1983.

[Buchanan, 1987] Bruce Buchanan. Artificial intelligence as an experimental science. Technical Report KSL 87-03, Knowledge Systems Laboratory, Stanford University, January 1987.

[Bylander and Chandrasekaran, 1987] T. Bylander and B. Chandrasekaran. Generic tasks in knowledge-based reasoning: The 'right' level of abstraction for knowledge acquisition. *International Journal of Man-Machine Studies*, 26(2):231–244, 1987.

[Chandrasekaran and Mittal, 1983] B. Chandrasekaran and S. Mittal. Conceptual representation of medical knowledge for diagnosis by computer: MDX and related systems. In M. Yovits, editor, *Advances in Computers*, pages 217–293. Academic Press, New York, 1983.

[Chandrasekaran *et al.*, 1982] B. Chandrasekaran, S. Mittal, and J. W. Smith. Reasoning with uncertain knowledge: the MDX approach. In *Proceedings of the Congress of American Medical Informatics Association*, pages 335–339, San Francisco, 1982.

[Chandrasekaran, 1983] B. Chandrasekaran. Towards a taxonomy of problem solving types. *AI Magazine*, 4(1):9–17, 1983.

[Chandrasekaran, 1986] B. Chandrasekaran. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, 1(3):23–30, Fall 1986.

[Chandrasekaran, 1987] B. Chandrasekaran. Towards a functional architecture for intelligence based on generic information processing tasks. In

Proceedings of the Tenth International Joint Conference on Artificial Intelligence, pages 1183–1192, 1987.

[Chapman and Agre, 1987] David Chapman and Philip E. Agre. Abstract reasoning as emergent from concrete activity. In M. P. Georgeff and A. L. Lansky, editors, *Reasoning About Actions and Plans, Proceedings of the 1986 Workshop at Timberline, Oregon*, pages 411–424, 1987.

[Clancey,] William J. Clancey. Acquiring, representing, and evaluating a competence model of diagnosis. KSL Memo 84-2, Stanford University, February 1984. To appear in Chi, Glaser, and Farr (Eds.), *Contributions to the Nature of Expertise*, in preparation.

[Clancey and Bock, 1988] William J. Clancey and Conrad Bock. Representing control knowledge as abstract tasks and metarules. In L. Bolc and M. J. Coombs, editors, *Expert System Applications*, pages 1–78. Springer-Verlag, New York, 1988. Previous version: report KSL-85-16, Stanford University.

[Clancey, 1983a] William J. Clancey. The advantages of abstract control knowledge in expert system design. In *Proceedings of the Third National Conference on Artificial Intelligence*, pages 74–78, Washington, D.C., August 1983.

[Clancey, 1983b] William J. Clancey. The epistemiology of a rule-based expert system: A framework for explanation. *Artificial Intelligence*, 20:215–251, 1983.

[Clancey, 1984a] W. J. Clancey. Details of the revised therapy algorithm. In B. G. Buchanan & E. H. Shortliffe, editor, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Reading MA, 1984.

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

[Clancey, 1985] William J. Clancey. Heuristic classification. *Artificial Intelligence*, 27:289–350, 1985.

[Clancey, 1986] William J. Clancey. From GUIDON to NEOMYCIN and HERACLES in twenty short lessons. *AI Magazine*, 7(3):40–60, 1986.

[Clancey, 1987] William J. Clancey. Viewing knowledge bases as qualitative models. Technical Report KSL-86-27, Computer Science Department, Stanford University, 1987. To appear in *IEEE Expert*, 1988.

[Clancey, 1988] W. J. Clancey. Acquiring, representing, and evaluating a competence model of diagnosis. In Chi, Glaser, and Farr, editors, *Contributions to the Nature of Expertise*. Lawrence Erlbaum, Hillsdale N.J., 1988. previously published as KSL Memo 84-2, Standford University, February, 1984.

[Clancey, forthcoming] William J. Clancey. Representing control knowledge as abstract tasks and metarules. In M. Coombs and L. Bolc, editors, *Computer Expert Systems*. Springer-Verlag, forthcoming. Also KSL 85–16, Stanford University.

[Cohen and Day, 1987] Paul R. Cohen and David S. Day. Planning in complex real-world time-dependent domains. In *Proceedings of the DARPA Knowledge-Based Planning Workshop*, Austin, Texas, December 1987.

[Cohen and Day, 1988] Paul R. Cohen and David S. Day. The centrality of autonomous agents in theories of action under uncertainty. Eksl technical report, University of Massachusetts,. January 1988. To appear in the *International Journal for Approximate Reasoning.*

[Cohen and Feigenbaum, 1982] Paul R. Cohen and Edward Feigenbaum. *The Handbook of Artificial Intelligence, Volume III.* William Kaufmann, Inc., Los Altos, CA, 1982.

[Cohen and Gruber, 1985] Paul R. Cohen and Thomas R. Gruber. Reasoning about uncertainty: A knowledge representation perspective. *Pergamon Infotech State of the Art Report*, 1985. Also, COINS Technical Report 85-24, Department of Computer and Information Science, University of Massachusetts.

[Cohen and Howe, 1988a] Paul Cohen and Adele E. Howe. Toward AI research methodology: Three case studies in evaluation. Technical Report COINS 88-31, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, 1988. To appear in IEEE Transactions on Systems, Man & Cybernetics.

[Cohen and Howe, 1988b] Paul R. Cohen and Adele E. Howe. How evaluation guides AI research. *AI Magazine*, 9(4):35–43, 1988.

[Cohen and Howe, 1988c] Paul R. Cohen and Adele E. Howe. Is there a method to our madness?: Case studies in evaluation. EKSL technical report, University of Massachusetts, 1988. To appear in *IEEE Systems, Man, and Cybernetics.*

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

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

[Cohen and Loiselle, 1988] Paul R. Cohen and Cynthia L. Loiselle. Beyond ISA: Structures for plausible inference in semantic networks. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, Saint Paul, Minnesota, 1988.

[Cohen and Stanhope, 1986] Paul R. Cohen and Philip Stanhope. Finding research funds with the grant system. In *Proceedings of the Sixth International Workshop on Expert Systems and Their Applications*, 1986. April 28-30, 1986, Avignon, France.

[Cohen *et al.*, 1985] 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.

[Cohen *et al.*, 1987a] Paul R. Cohen, David S. Day, Jeff Delisio, Michael Greenberg, Rick Kjeldsen, Dan Suthers, and Paul Berman. Management of uncertainty in medicine. *International Journal of Approximate Reasoning*, 1(1):103–116, 1987.

[Cohen *et al.*, 1987b] Paul R. Cohen, Michael Greenberg, and Jeff Delisio. MU: A development environment for prospective reasoning systems. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 783–788, Seattle, Washington, July 1987.

[Cohen *et al.*, 1987c] Paul R. Cohen, Glenn Shafer, and Prakash P. Shenoy. Modifiable combining functions. *AI EDAM*, 1(1):47–85, 1987.

[Cohen *et al.*, 1988] Paul R. Cohen, Jefferson L. DeLisio, and David Hart. A declarative representation of control knowledge. *IEEE Transactions on Systems, Man and Cybernetics*, 1988.

[Cohen *et al.*, 1989] P. R. Cohen, Michael L. Greenberg, David M. Hart, and Adele E. Howe. Trial by fire: Understanding the design requirements for agents in complex environments. Technical report, Coins Dept., University of Masschusetts, 1989. TR 89-61.

[Cohen, 1986] Paul R. Cohen. Managing uncertainty. In *The Third Conference on Artificial Intelligence Applications*, Orlando, Florida, February, 1987 1986. Department of Computer and Information Science, University of Massachusetts.

[Cohen, 1987a] Paul R. Cohen. The control of reasoning under uncertainty: A discussion of some programs. *The Knowledge Engineering Review*, 2(1), March 1987.

[Cohen, 1987b] Paul R. Cohen. Steps toward programs that manage uncertainty. In *Third Workshop on Uncertainty in Artificial Intelligence*, pages 372–379. American Association for Artificial Intelligence, July 1987.

[Cohen, 1988] Paul R. Cohen. An adaptive planner for real-time, uncertain environments. Technical report, Department of Computer and Information Science, 1988. Unpublished proposal.

[Cohen, 1989] P R. Cohen. Why knowledge systems research is in trouble and what we can do about it. Technical report, COINS Dept., University of Massachusetts, 1989.

[Collins *et al.*, 1975] A. Collins, E. Warnock, N. Aiello, and M. Miller. Reasoning from incomplete knowledge. In D. G. Bobrow and A. Collins, editors, *Representation and Understanding*, pages 383–415. Academic Press, New York, 1975.

[Collins, 1978] A. Collins. Fragments of a theory of human plausible reasoning. In D. Waltz, editor, *Theoretical Issues in Natural Language Processing*. University of Illinois, Urbana, IL, 1978.

[Croft *et al.*, 1988] W. Bruce Croft, T. J. Lucia, and P. R. Cohen. Retrieving documents by plausible inference: A preliminary study. Technical report, University of Massachusetts, Department of Computer and Information Science, Amherst, MA. 01003, 1988. COINS Technical Report 88-16.

[Dacus, 1987] Jim R. Dacus. Knowledge-based planning in advanced tactical aircraft. In *Proceedings of the DARPA Knowledge-Based Planning Workshop*, Austin, Texas, December 1987.

[Daily *et al.*, 1987] M. Daily, J. Harris, D. Deirsey, K. Olin, D. Payton, K Reiser, J. Rosenblatt, D. Tseng, and V. Wong. Autonomous cross-country navigation with the ALV. In *Proceedings of the DARPA Knowledge-Based Planning Workshop*, Austin, Texas, December 1987.

[DAR, 1987] DARPA. *Knowledge-Based Planning Workshop*, Austin, Texas, July 1987.

[Davis, 1976] Randall Davis. *Applications of meta-level knowledge to the construction, maintenance, and use of large knowledge bases.* PhD thesis, Computer Science Department, Stanford University, 1976. Reprinted in R. Davis and D. B. Lenat (Eds.), *Knowledge-Based Systems in Artificial Intelligence*, New York: McGraw-Hill, 1982.

[Davis, 1982] R. Davis. Applications of meta-level knowledge to the construction, maintenance, and use of large knowledge bases. In R. Davis and D. B. Lenat, editors, *Knowledge-Based Systems in Artificial Intelligence.* McGraw-Hill, New York, 1982. Doctoral dissertation, Computer Science Department, Stanford University.

[Dawson *et al.*, 1986] 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.

[Day, 1987a] David S. Day. JANUS: An architecture for integrating automatic and controlled problem solving. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, pages 655–662, Seattle, Washington, 1987.

[Day, 1987b] David S. Day. Towards integrating automatic and controlled problem solving. In *Proceedings of the First International Conference on Neural Networks.* Institute of Electronic and Electrical Engineers, 1987.

[Dean, 1987a] Thomas Dean. Large-scale temporal data bases for planning in complex domains. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, 1987.

[Dean, 1987b] Thomas Dean. Planning, execution and control. In *Proceedings of the DARPA Knowledge-Based Planning Workshop*, Austin, Texas, December 1987.

[DeJong and Mooney, 1986] Gerald DeJong and Raymond Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1(2), 1986.

[Dietterich and Michalski, 1983] T. G. Dietterich and R.S. Michalski. A comparative review of selected methods for learning from examples. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 41–81. Tioga Press, Palo Alto, CA, 1983.

[Dietterich and Michalski, 1986] T. G. Dietterich and R. S. Michalski. Learning to predict sequences. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach, Volume II*, volume 2, pages 63–106. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1986.

[Dietterich, 1982] T. G. Dietterich. Learning and inductive inference. In Paul R. Cohen and Edward Feigenbaum, editors, *The Handbook of Artificial Intelligence, Volume III*. William Kaufmann, Inc., 1982.

[Dixon *et al.*, 1984] 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.

[Durfee and Lesser, 1987] E. H. Durfee and V. R. Lesser. Incremental planning to control a time-constrained, blackboard-based planner. *IEEE Transactions on Aerospace and Electronic Systems*, 1987. To appear.

[Durfee, 1987] Edmund H. Durfee. A unified approach to dynamic coordination: Planning actions and interactions in a distributed problem solving network. Ph.d. dissertation, University of Massachusetts, Amherst, Massachusetts, September 1987.

[E. L. Hutchins and Norman, 1986] J. D. Hollan E. L. Hutchins and D. A. Norman. Direct minipulation interfaces. In D. A. Norman and S. W. Draper, editors, *User Centered System Design*. Lawrence Erlbaum Associates, Hillsdale, N. J., 1986.

[Erman and Lesser, 1975] L. Erman and V.R. Lesser. A multi-level organization for problem solving using many diverse, cooperating sources of knowledge. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, Stanford, California, 1975.

[Erman *et al.*, 1980] 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.

[Erman *et al.*, 1984] Lee D. Erman, A. Carlisle Scott, and Phillip London. Separating and integrating control in a rule-based tool. In *Proceedings of the IEEE Workshop on Principles of Knowledge-base Systems*, pages 37–43, Denver, Colorado, December 1984.

[Eshelman, 1987] L. Eshelman. MOLE: A knowledge acquisition tool that buries certainty factors. In *Proceedings of the Second AAAI Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff, Canada, October 1987. To appear in the *International Journal of Man-Machine Studies.*

[Eshelman, 1988] L. Eshelman. Mole: A knowledge-aquisition tool for cover-and differentiate systems. In S. Marcus, editor, *Automating Knowledge Acquisition for Expert Systems*. Kluwer Academic Publishers, Boston, MA, 1988.

[Fikes *et al.*, 1972] R. Fikes, P. Hart, and N. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4):251–288, 1972.

[Firby and Hanks, 1987] R. James Firby and Steve Hanks. A simulator for mobile robot planning. In *Proceedings of the DARPA Knowledge-Based Planning Workshop*, Austin, Texas, December 1987.

[Firby, 1987] R. James Firby. An investigation into reactive planning in complex domains. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 202–206, Seattle, Washington, 1987.

[Freiling and Alexander, 1984] M. J. Freiling and J. H. Alexander. Diagrams and grammars: Tools for mass producing expert systems. In *Proceedings on the First Conference on Artificial Intelligence Applications*, pages 537–543, Denver, Colorado, 1984. IEEE Computer Society Press.

[Gale, 1987] W. A. Gale. Knowledge-based knowledge acquisition for a statistical consulting system. *International Journal of Man-Machine Studies*, 13:81–116, 1987.

[Garvey *et al.*, 1987] Alan Garvey, Craig Cornelius, and Barbara Hayes-Roth. Computational costs versus benefits of control reasoning. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 110–115, Seattle, Washington, 1987.

[Gaschnig *et al.*, 1983] 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.

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

[Georgeff and Lansky, 1987] Michael P. Georgeff and Amy L. Lansky, editors. *The 1986 Workshop on Reasoning about Actions and Plans*, Timberline, Oregon, 1987.

[Golding *et al.*, 1987] Andrew Golding, Paul S. Rosenbloom, and John E. Laird. Learning general search control from outside guidance. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 334–337, Milan, Italy, 1987.

[Gruber and Cohen, 1987a] Thomas R. Gruber and Paul R. Cohen. Design for acquisition: principles of knowledge system design to facilitate knowledge acquisition. *International Journal of Man-Machine Studies*, 26(2):143–159, 1987.

[Gruber and Cohen, 1987b] Thomas R. Gruber and Paul R. Cohen. Knowledge engineering tools at the architecture level. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 100–103, Milan, Italy, 1987.

[Gruber and Cohen, 1987c] Thomas R. Gruber and Paul R. Cohen. Principles of design for knowledge acquisition. In *Proceedings of the Third IEEE Artificial Intelligence Applications Conference*, Orlando, Florida, February 1987, 1987.

[Gruber, 1987] Thomas R. Gruber. Acquiring strategic knowledge from experts. In *Proceedings of the Second AAAI Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff, Canada, October 1987. To appear in the *International Journal of Studies*.

[Gruber, 1988a] Thomas R. Gruber. Acquiring strategic knowledge from experts. *International Journal of Man-Machine Studies*, 1988. Forthcoming.

[Gruber, 1988b] Thomas R. Gruber. A method for acquiring strategic knowledge. Technical report, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, 1988.

[Gruber, 1989] Thomas R. Gruber. *The Acquisition of Strategic Knowledge*. Academic Press, Boston, 1989. based on doctoral dissertation, Department of Computer and Information Science, University of Massachusetts.

[Hammond, 1986] Kristian Hammond. The use of remindings in planning. In *Proceedings of the Cognitive Science Society*, 1986.

[Hammond, 1989] K. J. Hammond. *Cased-Based Planning: Viewing Planning as a Memory Task*. Academic Press, Boston, 1989. Based on doctoral dissertation, Computer Science Department, Yale University.

[Hanks, 1987] Steve Hanks. Temporal reasoning about uncertain worlds. In *Third Workshop on Uncertainty in Artificial Intelligence*, pages 114–122. American Association for Artificial Intelligence, July 1987.

[Hannan and Politakis, 1985] J. Hannan and P. Politakis. Essa: An approach to acquiring decision rules for diagnostic expert systems. In *Proceedings of the Second Conference on Artificial Intelligence Applications*, pages 520–525, 1985.

[Hayes-Roth and Hayes-Roth, 1979] Barbara Hayes-Roth and Frederick Hayes-Roth. A cognitive model of planning. *Cognitive Science*, 3:275–310, 1979.

[Hayes-Roth *et al.*, 1986] B. Hayes-Roth, A. Garvey, M.V. Johnson, and M. Hewett. A layered environment for reasoning about action. Technical Report KSL 86–38, Computer Science Department, Stanford University, April 1986.

[Hayes-Roth *et al.*, 1987] Barbara Hayes-Roth, Alan Garvey, M.Vaughan Johnson, and Michael Hewett. A modular and layered environment for reasoning about action. Technical Report KSL 86–38, Computer Science Department, Stanford University, April 1987.

[Hayes-Roth, 1984] F. Hayes-Roth. Knowledge-based expert systems–the state of the art in the us. *Pergamon Infotech State of the Art Report*, 1984.

[Hayes-Roth, 1985] B. Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26:251–321, 1985.

[Hayes-Roth, 1987] Barbara Hayes-Roth. Dynamic control planning in adaptive intelligent systems. In *Proceedings of the DARPA Knowledge-Based Planning Workshop*, Austin, Texas, December 1987.

[Hays, 1973] Willim L. Hays. *Statistics for the Social Sciences*. Holt, Rinehart, and Winston, second edition, 1973.

[Hendler and Sanborn, 1987] James A. Hendler and James C. Sanborn. A model of reaction for planning in dynamic environments. In *Proceedings of the DARPA Knowledge-Based Planning Workshop*, Austin, Texas, December 1987.

[Henrion and Cooley, 1987] 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*, Seattle, Washington, 1987.

[Herman and Albus, 1987] Martin Herman and James S. Albus. Real-time hierarchical planning for multiple mobile robots. In *Proceedings of the DARPA Knowledge-Based Planning Workshop*, Austin, Texas, December 1987.

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

[Howe and Cohen, 1987] Adele E. Howe and Paul R. Cohen. A typology for constructing decisions. In *Proceedings of the Sixth Annual IEEE Conference on Computers and Communication*, Scottsdale, AZ, February 1987.

[Howe and Cohen, 1988] Adele E. Howe and Paul R. Cohen. Steps toward automating decision-making. Under review, Cognitive Science, 1988.

[Howe *et al.*, 1986] 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.

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

[Johnson and Tomlinson, 1988] N. E. Johnson and C. M. Tomlinson. Knowledge representation for knowledge elicitation. In *Proceedings of the third AAAI Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff, Canada, November 1988. Calgary, Alberta: SRDG Publications, Department of Computer Science, University of Calgary.

[Jordan, 1986] Michael I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Cognitive Science Society*, 1986.

[Kaelbling, 1987] Leslie Pack Kaelbling. An architecture for intelligent reactive systems. In M. P. Georgeff and A. L. Lansky, editors, *Reasoning About Actions and Plans, Proceedings of the 1986 Workshop at Timberline, Oregon*, pages 411–424, 1987.

[Kay, 1984] A. Kay. Computer software. *Scientific American*, 251(3):52–59, September 1984.

[Keeney and H.Raiffe, 1976] R. L. Keeney and H.Raiffe. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. John Wiley and Sons, 1976.

[Keller, 1988] Richard M. Keller. Defining operationality for explanation-based learning. *Artificial Intelligence*, 35(2):227–241, 1988.

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

[Kjeldsen and Cohen, 1987] Rick Kjeldsen and Paul R. Cohen. The evolution and performance of the GRANT system. *IEEE Expert*, 2(2):73–79, Spring 1987.

[Klinker, 1988] G. Klinker. Knack: Sample-driven knowledge acquisition for reporting systems. In *Automating Knowledge Acquisition for Expert Systems*. Kluwer Academic, Boston, 1988.

[Korf, 1987] Richard Korf. Real-time heuristic search: First results. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 133–138, Seattle, Washington, 1987.

[Laird *et al.*, 1986] John E. Laird, Paul S. Rosenbloom, and Allen Newell. Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning*, 1:11–46, 1986.

[Laird *et al.*, 1987] John E. Laird, Allen Newell, and Paul S. Rosenbloom. SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.

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

[Lenat and Brown, 1984] Douglas B. Lenat and John Seeley Brown. Why AM and EURISKO appear to work. *Artificial Intelligence*, 23(3), 1984.

[Lenat and Feigenbaum, 1987] D. B. Lenat and E. Feigenbaum. On the thresholds of knowledge. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 1173–1182, Milan, Italy, 1987.

[Lenat *et al.*, 1986] D. B. Lenat, M. Prakash, and M. Shepherd. CYC: Using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks. *AI Magazine*, 6(4):65–85, Winter 1986.

[Lenat, 1976] 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.

[Lesser *et al.*, 1988] Victor R. Lesser, Edmund H. Durfee, and Jasmina Pavlin. Approximate processing in real-time problem solving. *AI Magazine*, pages 49–61, Spring 1988.

[Luhrs and Nowicki, 1987] Richard A. Luhrs and Anthony R. Nowicki. Real-time dynamic planning for autonomous vehicles. In *Proceedings of the DARPA Knowledge-Based Planning Workshop*, Austin, Texas, December 1987.

[M. A. Musen and Shortliffe, 1986] L. M. Fagen M. A. Musen and E. H. Shortliffe. Graphical specification of procedural knowledge for an expert system. In *Proceedings of the1986 IEEE Computer Society Workshop on Visual Languages*, pages 167–178, Dallas, Texas, 1986.

[Marcus *et al.*, 1985] S. Marcus, J. McDermott, and T. Wang. Knowledge acquisition for constructive systems. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 637–639, Los Angeles, CA, August 1985.

[Marcus, 1987] Sandra Marcus. Taking backtracking with a grain of SALT. *International Journal of Man-Machine Studies*, 26(4):383–398, 1987.

[Marcus, 1988] S. Marcus. Salt: A knowledge acquisition tool for propose-and-refine systems. In S. Marcus, editor, *Automating Knowlege Acquisition for Expert Systems*. Kluwer Academic, Boston, 1988.

[McDermott, 1982] Drew McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[McDermott, 1987] Drew V. McDermott. DARPA-sponsered planning research: Report and prospectus. YALE/CSD/RR 522, Yale University, Department of Computer Science, March 1987.

[McDermott, 1988] J. McDermott. Preliminary steps toward a taxonomy of problem-solving methods. In S. Marcus, editor, *Automating Knowledge Acquistion for Expert Systems*. Kluwer Academic, 1988.

[Michalski *et al.*, 1986] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine Learning: An Artificial Intelligence Approach*, volume II. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1986.

[Minton and Carbonell, 1987] Steven Minton and Jaime G. Carbonell. Strategies for learning search control rules: An explanation-based approach. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 228–235, Milan, Italy, 1987.

[Mitchell *et al.*, 1983] T. M. Mitchell, P. E. Utgoff, and R. B. Banerji. Learning by experimentation: Acquiring and refining problem-solving heuristics. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach, Volume I.* Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1983.

[Mitchell *et al.*, 1985] T.M. Mitchell, S. Mahadevan, and L. Steinberg. LEAP: A learning apprentice for vlsi design. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 1985.

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

[Mitchell, 1977] 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.

[Mitchell, 1982] Tom M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[Morik, in press] Katharina Morik. Sloppy modeling. In Katharina Morik, editor, *Knowledge Representation and Organization in Machine Learning*. Springer-Verlag, Berlin, in press.

[Mostow, 1983] D. Jack Mostow. Machine transformation of advice into a heuristic search procedure. In R.S. Michalski, J.G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 243–306. Tioga Press, Palo Alto, CA, 1983.

[Musen *et al.*, 1987] Mark A. Musen, Lawrence M. Fagan, D. M. Combs, and Edward H. Shortliffe. Use of a domain model to drive an interactive knowledge editing tool. *International Journal of Man-Machine Studies*, 26(1):105–121, 1987.

[Musen, 1989] M. A. Musen. *Automated Generation of Model-based Knowledge-Acquisition Tools*. Pitman, London, 1989. Based on doctoral dissertation, Computer Science Department, Stanford University.

[Neches *et al.*, 1985] R. Neches, W.R. Swartout, and J. Moore. Enhanced maintenance and explanation of expert systems through explicit models of their development. *Transactions on Software Engineering*, SE-11(11):1337–1351, 1985.

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

[Newell and Simon, 1972] A. Newell and H.A. Simon. *Human Problem Solving*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1972.

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

[Newell, 1973] 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.

[Newell, 1975] 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.

[Newell, 1982] A. Newell. The knowledge level. *Artificial Intelligence*, 18:87–127, 1982.

[Nisbett and Wilson, 1977] R.E. Nisbett and T.D. Wilson. Telling more than we can know: Verbal reports on mental processes. *Psychological Review*, 84:231–259, 1977.

[Orelup *et al.*, 1988] Mark E. Orelup, John R. Dixon, Paul R. Cohen, and Melvin K. Simmons. Dominic ii: Meta-level control in iterative redesign. In *AAAI88*, pages 25–30, 1988.

[Orelup, 1987] Mark Ferral Orelup. Meta-control in domain-independent design by iterative redesign. Master's thesis, Mechanical Engineering Department, University of Massachusetts, September 1987.

[P R. Cohen and Howe, 1989] D. Hart P R. Cohen, M. L. Greenberg and A. E. Howe. An introduction to phoenix, the eksl fire-fighting system. Technical report, COINS Dept., University of Massachusetts, 1989.

[Pardee and Hayes-Roth, 1987] William J. Pardee and Barbara Hayes-Roth. Intelligent real time control of material processing. Technical report, Rockwell International, Science Center, Palo Alto Laboratory, February 1987.

[Pauker and Kassirer, 1981] S. G. Pauker and Jerome P. Kassirer. Clinical decision analysis by personal computer. *Archives of Internal Medicine*, 141:1831–1837, 1981.

[Pauker *et al.*, 1976] S.G. Pauker, A. Gorry, J.P. Kassirer, and W.B. Schwartz. Towards the simulation of clinical cognition: Taking a present illness by computer. *American Journal of Medicine*, 60:981–996, 1976.

[Polya, 1954] G. Polya. *Patterns of Plausible Inference*. Princeton University Press, Princeton, New Jersey, 1954.

[Quinlan, 1986] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

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

[Rothenberg *et al.*, 1987] 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.

[Rumelhart and Norman, 1982] D.E. Rumelhart and D.A. Norman. Simulating a skilled typist: A study of skilled cognitive-motor performance. *Cognitive Science*, 6:1–36, 1982.

[Sacerdoti, 1974] E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, September 1974.

[Sacerdoti, 1975] E.D. Sacerdoti. *A Structure for Plans and Behavior.* American Elsevier: New York, NY, 1975.

[Schneider and Shiffrin, 1984] Schneider and Shiffrin. Controlled and automatic human information processing: I. detection, search and attention. *Psychological Review*, 1984.

[Schneider, 1985] Walter Schneider. Toward a model of attention and the development of automatic processing. In Posner and Marin, editors, *Attention and Performance XI.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1985.

[Shachter and Heckerman, 1987] R. D. Shachter and D. E. Heckerman. Thinking backward for knowledge acquisition. *AI Magazine*, 8(3):55–61, 1987.

[Shaw and Gaines, 1987] M. Shaw and B. Gaines. Techniques for knowledge acquisition and transfer. *International Journal of Man-Machine Studies*, 27(1), 1987.

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

[Silver, 1986] Bernard Silver. *Meta-level Inference: Representing and Learning Control Information in Artificial Intelligence.* North-Holland, New York, 1986.

[Simon, 1981] H. A. Simon. *The Sciences of the Artificial.* The MIT Press, Cambridge, MA, 1981.

[Sussman, 1975] G. J. Sussman. *A Computer Model of Skill Acquisition.* American Elsevier, New York, NY, 1975.

[Sutton and Barto, 1981] Richard S. Sutton and Andrew G. Barto. Toward a modern theory of adaptive networks: Expectation and prediction. *Psychological Review*, 88(2):135–170, 1981.

[Swartout, 1983] W. Swartout. XPLAIN: A system for creating and explaining expert consulting systems. *Artificial Intelligence*, 11:115–144, 1983.

[Swartout, 1987] William Swartout. Summary report on DARPA Santa Cruz workshop on planning. In *Proceedings of the DARPA Knowledge-Based Planning Workshop*, Austin, Texas, December 1987.

[Sycara, 1987] Katia Sycara. Planning for negotiation: A case-based approach. In *Proceedings of the DARPA Knowledge-Based Planning Workshop*, Austin, Texas, December 1987.

[Tate, 1974] Austin Tate. INTERPLAN: A plan generation system that can deal with interactions between goals. Machine Intelligence Research Unit Memo MIP-1-109, University of Edinburgh, Edinburgh, December 1974.

[Tate, 1977] A. Tate. Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 888–893. Tiblisi, Georgia, USSR, 1977.

[Tong and Appelbaum, 1987] Richard M. Tong and Lee A. Appelbaum. Problem structure and evidential reasoning. In *Third Workshop on Uncertainty in Artificial Intelligence*, pages 313–319. American Association for Artificial Intelligence, July 1987.

[Tu *et al.*, 1989] S. W. Tu, M. G. Kahn, M. G. Musen, M. A. Ferguson, E. H. Shortliffe, and L. M. Fagen. Episodic monitoring of time-oriented data for heuristic skeletal-plan refinement. *Communications of the ACM*, 1989. in press.

[Utgoff, 1986] Paul Utgoff. *Machine Learning of Inductive Bias.* Kluwer Academic, Norwell, MA, 1986.

[Vere, 1981] Stephen Vere. Planning in time: Windows and durations for activites and goals. Technical report, Jet Propulsion Laboratory, Pasadena, California, 1981.

[Waldinger, 1977] Richard Waldinger. Achieving several goals simultaneously. In E. W. Elcock and D. Michie, editors, *Machine Intelligence, Volume 8*. Halstead/Wiley, New York, NY, 1977.

[Waltz, 1975] D. Waltz. Generating semantic descriptions from drawings of scenes with shadows. In P. Winston, editor, *The Psychology of Computer Vision*, pages 19–92. McGraw-Hill, New York, 1975.

[Waterman, 1970] D. A. Waterman. Generalization learning techniques for automating the learning of heuristics. *Artificial Intelligence*, 1:121–170, 1970.

[Weiss *et al.*, 1977] S. Weiss, C. Kulikowski, and A. Safir. A model-based consultation system for the long-term management of glaucoma. In *Proceedings IJCAI 5*, pages 826–832, 1977.

[Weiss *et al.*, 1978] S.M. Weiss, C.A. Kulikowski, S. Amarel, and A. Safir. A model-based method for computer-aided medical decision-making. *Artificial Intelligence*, 11:145–172, 1978.

[Wellman and Heckerman, 1987] Michael P. Wellman and David E. Heckerman. The role of calculi in uncertain reasoning. In *Third Workshop on Uncertainty in Artificial Intelligence*, pages 321–331. American Association for Artificial Intelligence, July 1987.

[Wellman, 1987] Michael P. Wellman. Qualitative probabilistic networks for planning under uncerainty. In John F. Lemmer, editor, *Uncertainty in Artificial Intelligence*. 1987.

[Wilkins, 1985] David E. Wilkins. Recovering from execution errors in SIPE. Technical report, SRI International, 1985. Technical Note No. 346, Artificial Intelligence Center, Computer Science and Technology Center.

[Winograd and Flores, 1987] Terry Winograd and Fernando Flores. *Understanding computers and cognition*. Addison-Wesley, Reading, MA, 1987.

[Winston, 1975] P.H. Winston, editor. *The Psychology of Computer Vision*. McGraw Hill: New York, NY, 1975.

[Winterfeldt and Edwards, 1986] Detlof Von Winterfeldt and Ward Edwards. *Decision Analysis and Behavioral Research.* Cambridge University Press, Cambridge, 1986.

DISTRIBUTION LIST

addresses	number of copies
RADC/COES ATTN: Chuian-Chuian Hwong Griffiss AFB NY 13441-5700	5
Lederle Graduate Research Center University of Massachusetts P.O. Box 571 Amherst MA 01003	5
RADC/DOVL Technical Library Griffiss AFB NY 13441-5700	1
Administrator Defense Technical Info Center DTIC-FDA Cameron Station Building 5 Alexandria VA 22304-6145	5
Defense Advanced Research Projects Agency 1400 Wilson Blvd Arlington VA 22209-2308	2
AFCSA/SAMI ATTN: Miss Griffin 10363 Pentagon Washington DC 20330-5425	1
HQ USAF/SCTT Pentagon Washington DC 20330-5190	1
SAF/AQSC Pentagon 4D-267 Washington DC 20330-1000	1

Director, Information Systems 1
OASD (C3I)
Room 3E187
Pentagon
Washington DC 20301-3040

Naval Warfare Assessment Center 1
GIDEP Operations Center
ATTN: Mr. E. Richards
Code 3061
Corona CA 91720

HQ AFSC/XTKT 1
Andrews AFB DC 20334-5000

HQ AFSC/XTS 1
Andrews AFB MD 20334-5000

HQ AFSC/XRK 1
Andrews AFB MD 20334-5000

HQ SAC/SCPT 1
OFFUTT AFB NE 68113-5001

DTESA/RQE 1
ATTN: Mr. Larry G.McManus
Kirtland AFB NM 87117-5000

HQ TAC/DRIY 1
ATTN: Mr. Westerman
Langley AFB VA 23665-5001

HQ TAC/DOA 1
Langley AFB VA 23665-5001

HQ TAC/DRCA
Langley AFB VA 23665-5001 1

ASD/AFALC/AXAE
ATTN: W. H. Dungey
Wright-Patterson AFB OH 45433-6533 1

WRDC/AAAI
Wright-Patterson AFB OH 45433-6533 1

AFIT/LDEE
Building 640, Area B
Wright-Patterson AFB OH 45433-6583 1

WRDC/MLTE
Wright-Patterson AFB OH 45433 1

WRDC/FIES/SURVIAC
Wright-Patterson AFB OH 45433 1

AAMRL/HE
Wright-Patterson AFB OH 45433-6573 1

2750 ABW/SSLT
Building 262
Post 11S
Wright-Patterson AFB OH 45433 1

AFHRL/OTS
Williams AFB AZ 85240-6457 1

AUL/LSE 1
Maxwell AFB AL 36112-5564

HQ Air Force SPACECOM/XPYS 1
ATTN: Dr. William R. Matoush
Peterson AFB CO 80914-5001

Defense Communications Engr Center 1
Technical Library
1860 Wiehle Avenue
Reston VA 22090-5500

C3 Division Development Center 2
Marine Corps
Development & Education Command
Code DIOA
Quantico VA 22134-5080

US Army Strategic Defense Command 1
DASD-H-MPL
PO Box 1500
Huntsville AL 35807-3801

Commanding Officer 1
Naval Avionics Center
Library
D/765
Indianapolis IN 46219-2189

Commanding Officer 1
Naval Ocean Systems Center
Technical Library
Code 96429
San Diego CA 92152-5000

Commanding Officer 1
Naval Weapons Center
Technical Library
Code 3433
China Lake CA 93555-6001

Superintendent 1
Naval Post Graduate School
Code 1424
Monterey CA 93943-5000

Commanding Officer 2
Naval Research Laboratory
Code 2627
Washington DC 20375-5000

Space & Naval Warfare Systems COMM 1
PMW 153-3DP
ATTN: R. Savarese
Washington DC 20363-5100

Commanding Officer 2
US Army Missile Command
Redstone Scientific Info Center
AMSMI-RD-CS-R (Documents)
Redstone Arsenal AL 35898-5241

Advisory Group on Electron Devices 2
Technical Info Coordinator
ATTN: Mr. John Hammond
201 Varick Street - Suite 1140
New York NY 10014

Los Alamos Scientific Laboratory 1
Report Librarian
ATTN: Mr. Dan Baca
PO Box 1663, MS-P364
Los Alamos NM 87545

Rand Corporation 1
Technical Library
ATTN: Ms. Doris Helfer
PO Box 2138
Santa Monica CA 90406-2138

USAG 1
ASH-PCA-CRT
Ft. Huachuca AZ 85613-6000

1839 EIG/EIET 1
ATTN: Mr. Kenneth W. Irby
Keesler AFB MS 39534-6348

JTFPO-TD 1
Director of Advanced Technology
ATTN: Dr. Raymond F. Freeman
1500 Planning Research Drive
McLean VA 22102

HQ ESC/CWPP
San Antonio TX 78243-5000 1

AFEWC/ESRI
San Antonio TX 78243-5000 3

485 EIG/EIR
ATTN: M Craft
Griffiss AFB NY 13441-6348 1

ESD/XTP
Hanscom AFB MA 01731-5000 1

ESD/ICP
Hanscom AFB MA 01731-5000 1

ESD/AVSE
Building 1704
Hanscom AFB MA 01731-5000 2

HQ ESD SYS-2
Hanscom AFB MA 01731-5000 1

Director
NSA/CSS
T513/TDL
ATTN: Mr. David Marjarum
Fort George G. Meade MD 20755-6000 1

Director
NSA/CSS
W166
Fort George G. Meade MD 20755-6000 1

Director 1
NSA/CSS
R24
Fort George G. Meade MD 20755-6000

Director 1
NSA/CSS
R21
9800 Savage Road
Fort George G. Meade MD 20755-6000

Director 1
NSA/CSS
DEFSMAC
ATTN: Mr. James E. Hillman
Fort George G. Meade MD 20755-6000

Director 1
NSA/CSS
R5
Fort George G. Meade MD 20755-6000

Director 1
NSA/CSS
R8
Fort George G. Meade MD 20755-6000

Director 1
NSA/CSS
S21
Fort George G. Meade MD 20755-6000

Director 1
NSA/CSS
W07
Fort George G. Meade MD 20755-6000

Director 1
NSA/CSS
W3
Fort George G. Meade MD 20755-6000

Director 2
NSA/CSS
R523
Fort George G. Meade MD 20755-6000

AFHRL/LRG 1
ATTN: Mr. M. Young
WPAFB OH 45433-6503

AAMRL/HEA 1
ATTN: Dr. B. Tsou
WPAFB OH 45433-6503

AAMRL/HED 1
ATTN: Maj M.R. McFarren
WPAFB OH 45433-6503

WRDC/KTD 1
ATTN: Dr. D. Hopper
WPAFB OH 45433-6503

AFIT/ENG 1
ATTN: Maj P. Amburn
WPAFB OH 45433-6053

CEETL-GL-VT 1
ATTN: Mr. T. Jorgensen
Ft Belvoir VA 22060-5546

Rensselaer Polytechnic Institute 1
ATTN: Dr. David Musser
Computer Science Dept
Troy NY 12180-3590

University of Texas at Austin 1
Dept of Computer Science
Taylor Hall 2.124
Austin TX 78712-1188

Weapons Laboratory 1
WL-SCD
ATTN: E. Carmona
Kirtland AFB NM 8717-6008

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic reliability/maintainability and compatibility.