MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

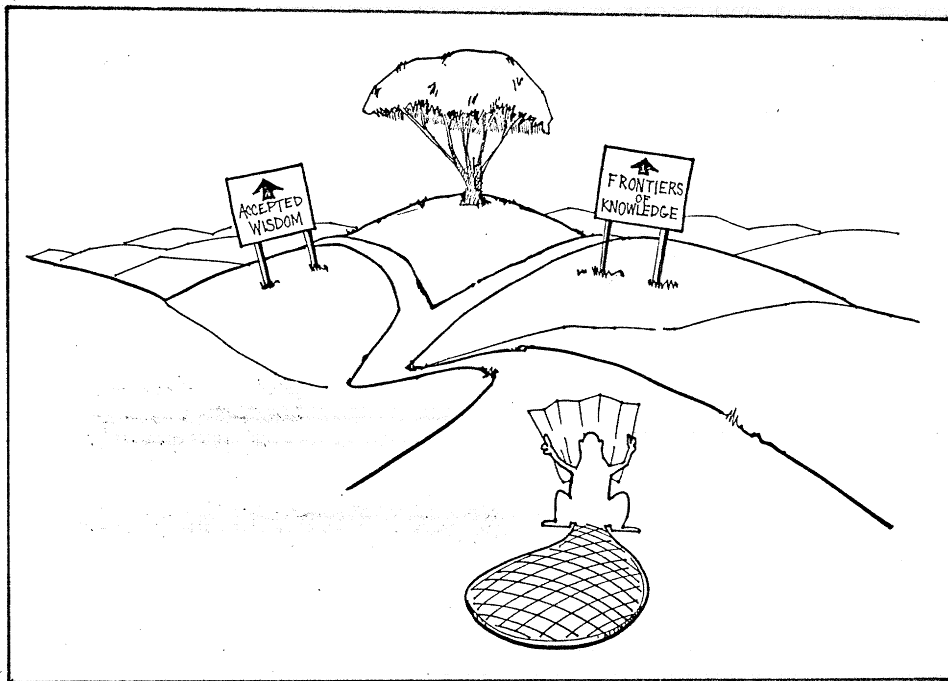
A.I. Memo No. 665 June 1982

EXPERT SYSTEMS: WHERE ARE WE?

AND WHERE DO WE GO FROM HERE?

Randall Davis

Abstract



Acknowledgements

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's A.I. research on hardware troubleshooting is provided by a research grant supplied to M.I.T. by the Digital Equipment Corporation.

© MASSACHUSETTS INSTITUTE OF TECHNOLOGY 1982

CONTENTS

1. INTRODUCTION 1

1.1 The Itinerary 1
1.2 A Glimpse Ahead 2
1.3 A Word About Terminology 2

2. THE NATURE OF EXPERTISE 3

3. IN THE LAND OF ACCEPTED WISDOM 7

3.1 Knowledge Engineering Homilies 7
3.2 Architectural Principles 8
3.3 The Current State of the Art 9
3.4 Success of the Accepted Wisdom 10
3.5 Calibrating the Technology 11
3.6 Stages of Development 13

4. TOWARD THE FRONTIERS 16

4.1 Where the AW Falls Short: The Architectural Principles Again 16
4.2 Where the AW Falls Short: Explanation and Knowledgexiairion 18

5. ACROSS THE FRONTIER 19

6. AN EXAMPLE 19

6.1 The Problem 20
6.2 Comparison with the Accepted Wisdom 25
6.3 The Example Continued 26
6.4 The Example: Summary 29

7. CHALLENGES 30

7.1 Diagnosis from Structure and Function 30
7.2 More General Issues 34

8. SUMMARY 35

9. REFERENCES 37

EXPERT SYSTEMS: WHERE ARE WE? AND WHERE DO WE GO FROM HERE?

Randall Davis †
MIT Artificial Intelligence Laboratory

INTRODUCTION

Work on Expert Systems has received extensive attention recently, prompting growing interest in a range of environments. Much has been made of the basic concept and of the rule-based system approach typically used to construct the programs. Perhaps this is a good time then to review what we know, assess the current prospects, and suggest directions appropriate for the next steps of basic research.

I'd like to do that today, and propose to do it by taking you on a journey of sorts, a metaphorical trip through the State of the Art of Expert Systems. We'll wander about the landscape, ranging from the familiar territory of the Land of Accepted Wisdom, to the vast unknowns at the Frontiers of Knowledge. I guarantee we'll all return safely, so come along...

The Itinerary

Our trip has three basic purposes in mind. I want first to assess and calibrate the Accepted Wisdom, second to understand its limitations, and third to characterize the nature of the research that will produce the next generation of systems. The talk is correspondingly divided into three parts. We begin with a tour through the Land of Accepted Wisdom, where we survey the field, trying to assess what's known about building and using these systems. What do we know and what can we do with that knowledge? Where has the Accepted Wisdom succeeded?

One important goal of this part of the journey will be to calibrate the current state of the art. Given what we know, what can we do and how quickly can we do it? Does building a system typically require several months or several years? Can we turn them out routinely or are we still hand-crafting them? Is the technology a laboratory curiosity or is it ready for mass production?

As the name suggests, any journey through the Land of Accepted Wisdom invariably means encountering familiar territory. But a review of fundamentals is important if we are to accomplish our second goal --- the careful determination of the nature and location of the Frontiers. We need to examine the Accepted Wisdom closely, to see where the ragged edges are and to see where experience may have worn holes in some of the cherished cliches. That examination forms the second part of our journey.

The shortcomings we find will lead us to our final goal, an understanding of which way to go next, as we head toward the Frontiers of Knowledge. Our vehicle will be a problem in equipment diagnosis that nicely characterizes the weaknesses of the existing technology and brings sharply into focus the challenges we face in building the next generation of systems.

---

† This is a revised and updated version of the author's Invited Lecture at the 7th IJCAI Conference, Vancouver, Canada, August 1981.

This article has benefited from careful readings by and suggestions from Mike Brady, Bruce Buchanan, Ed Feigenbaum, Jose Gonzalez, Pete Szolovits, and Pat Winston.

A Glimpse Ahead

It's standard practice in a talk to give away the punch-lines early on, so here's a preview:

```
                    The Central Issues

The Accepted Wisdom has provided an important foundation
     and a useful set of tools.
Those tools can be particularly powerful where performance
     is based largely on "compiled experience".
Additional varieties of knowledge are necessary beyond
     empirical associations.
Expertise is characterized by a range of behaviors.
New problem domains require additional mechanisms.
We need to reconcile the new directions with the advantages
     of the Accepted Wisdom.
```

Figure 1

In examining the intellectual foundation and the tools provided by the Accepted Wisdom, we'll see that they can be particularly powerful in domains where expertise is well described as "compiled experience". In clinical medicine, for example, a considerable amount of diagnostic expertise arises from exposure to numerous examples. Typically, relatively little of a physician's diagnostic capability is based on reasoning from detailed anatomical and physiological considerations. Instead, what makes a clinician an expert is the large collection of empirical associations he accumulates by dint of experience in the field. In domains of this sort, the existing set of tools can be useful and powerful.

But we have to press on beyond the conventional wisdom, because additional varieties of knowledge will be necessary in the next generation of systems. There are several reasons for this. First, as we'll see, expertise is characterized by a wide range of behaviors, of which problem solving performance --- the focus of current work on expert systems --- is really only the most evident. Experts do much more than simply solve the problem and it's not clear that empirical associations are capable of supporting those other forms of behavior.

Second, there are domains where problem solving clearly relies on more than compiled experience. Other varieties of knowledge are involved, knowledge of structure and causal models. To take advantage of the power they can provide, we're going to have to press beyond the existing technology.

We'll see that the desire to use those additional kinds of knowledge leads us off in directions that conflict with some of the advantages provided by the Accepted Wisdom. Hence one of the long term challenges we face is determining how to reconcile conflicts between the new directions and the advantages of the Accepted Wisdom.

A Word About Terminology

Let me focus for a moment on the phrases "empirical associations" and "causal models." They are, I think, more appropriate than "shallow" and "deep," words that have been used quite a bit recently in reference to much the same thing. The problem with "shallow" and "deep" is that they are simply labels whose technical content is at best obscure (i.e., they are shallow), while their overtones from nontechnical use make them far too evocative.

The technical issue is, what grounds exist for believing an inference like *If A and B then C*? If

the strongest argument we can make is of the sort, *Previously, when A and B held, C was also found to be true*, then the inference is justifiably characterizable as an *association* that grew from accumulated *empirical* observations.

If, on the other hand, we can argue that *If A and B are true, C follows, because of the way this device (or system, organ, etc.) works*, then we are basing our belief on an understanding of structure, function, or *causality*.

Such terms focus on the relevant issue and have the virtue of avoiding the unfortunate connotations of "shallow" (i.e., requiring little thought) and "deep" (i.e., profound) that carry over from everyday use. Clearly, for example, anyone who has a *shallow* system is missing something (and probably doing shoddy work). Anyone who's got a system based on *deep models*, on the other hand, is clearly doing something impressive.[1]

We should be wary too of the images suggested by the terms Accepted Wisdom and Frontiers. They are convenient abstractions and are rarely as clearcut and static as may be implied here. Nor is the population of each Land easily determined --- those who create the Accepted Wisdom at one moment are often found across the borders at the next. The metaphor is, nevertheless, a useful abstraction that allows us to categorize what we know and measure what we have yet to do.

THE NATURE OF EXPERTISE

Enough preliminaries. Let us begin by considering the nature of expertise. It includes a whole constellation of behaviors (Fig. 1). Problem solving is only the most obvious and while necessary, it is clearly insufficient. Would we be willing to call someone an expert if he could solve a problem, but was unable to explain the result, unable to learn anything new about the domain, unable to determine whether his expertise was relevant, etc.? I think not.

THE NATURE OF EXPERTISE

A Range of Behaviors:
- solve the problem
- explain the result
- learn
- restructure knowledge
- break rules
- determine relevance
- degrade gracefully

Figure 2

Work in expert systems has to date explored only the first three of these behaviors in any depth. First generation systems like DENDRAL [Lindsay et al., 1981] and MACSYMA [Macsyma group, 1974] focused solely on performance, while second generation systems began to explore behaviors like explanation (e.g., MYCIN [Shortliffe76], the Digitalis Advisor [Swartout 1981]) and knowledge acquisition (e.g., TEIRESIAS [Davis 1979], Version Spaces [Mitchell 1978]). Of these, performance is still the best understood; our efforts at explanation and knowledge acquisition have only scratched

1. *Deep* is a word like *theory* or *semantic* --- it implies all sorts of marvelous things. It's one thing to be able to say "I've got a theory," quite another to say, "I've got a semantic theory" but ah, those who can claim "I've got a deep semantic theory," they are truly blessed.

the surface.

By and large the other topics have been almost totally unexplored. What would it mean for example to restructure knowledge? One example particularly relevant for this audience came out of the so-called procedural vs. declarative controversy. We, as experts on knowledge representation, went to work structuring and restructuring knowledge, so that procedural became declarative, got turned back into procedural again, and so forth. I think it became clear after we went around that loop a couple of times that the problem is at least in part in the eye of the interpreter, but nevertheless, there we were, restructuring and reorganizing knowledge.

What about breaking rules? One of the most frustrating things to an apprentice is that he no sooner learns a rule than he discovers there are almost as many exceptions as there are rules. Experts clearly understand not merely the letter of the rule but the spirit as well.

What about determining relevance? Experts also know when they're beyond their depth. They know when a problem is outside their sphere of expertise and they know when the best answer they can give is to suggest asking someone else. Clearly none of our programs can do that as yet.

Experts also degrade gracefully. That is, as they get close to the boundaries of their expertise, they get less and less proficient at solving problems. But their skill decreases smoothly, rather than precipitously as do most of our programs.

To illustrate this I'd like to introduce a colleague of mine (Fig. 3), who'll be accompanying us on this journey. It's been suggested that the beaver, engineer of the animal kingdom and (not entirely by coincidence) mascot of MIT, would make a perfectly good expert system if performance alone were the issue. After all, he does a marvelous job of building a dam.

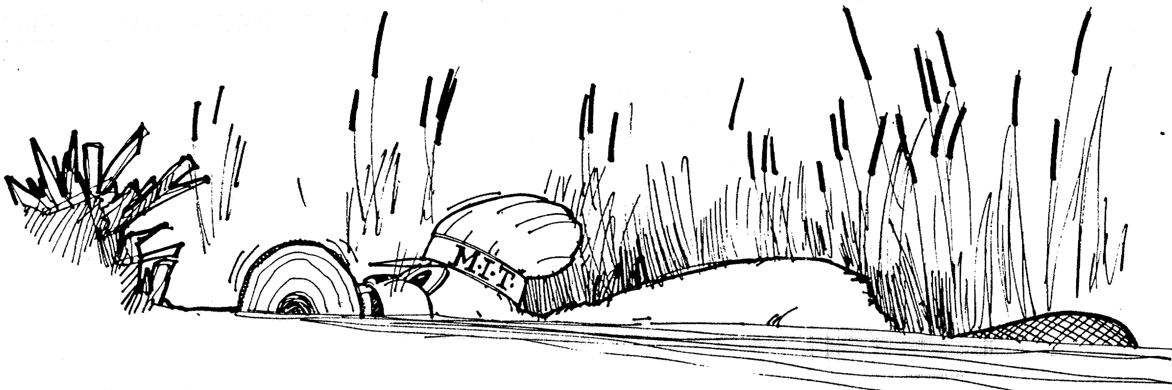


Figure 3 -- An Early Expert System

But performance isn't all there is. As I was saying a moment ago, experts can explain their behavior, know when to break rules, and can learn more about their domain. On that score, of course, our friend seems to fall short (Fig 4).

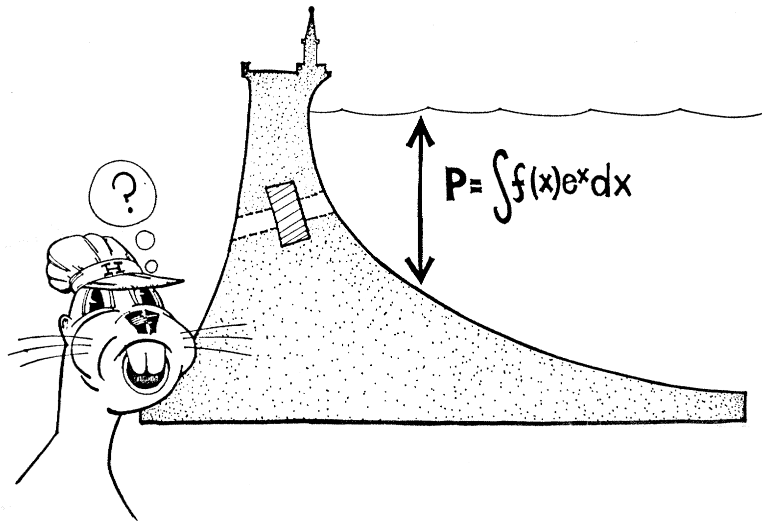


Figure 4 -- Outmoded by new technology

He will, nevertheless, accompany us as we head off down the road to see what lies ahead in the Land of Accepted Wisdom.

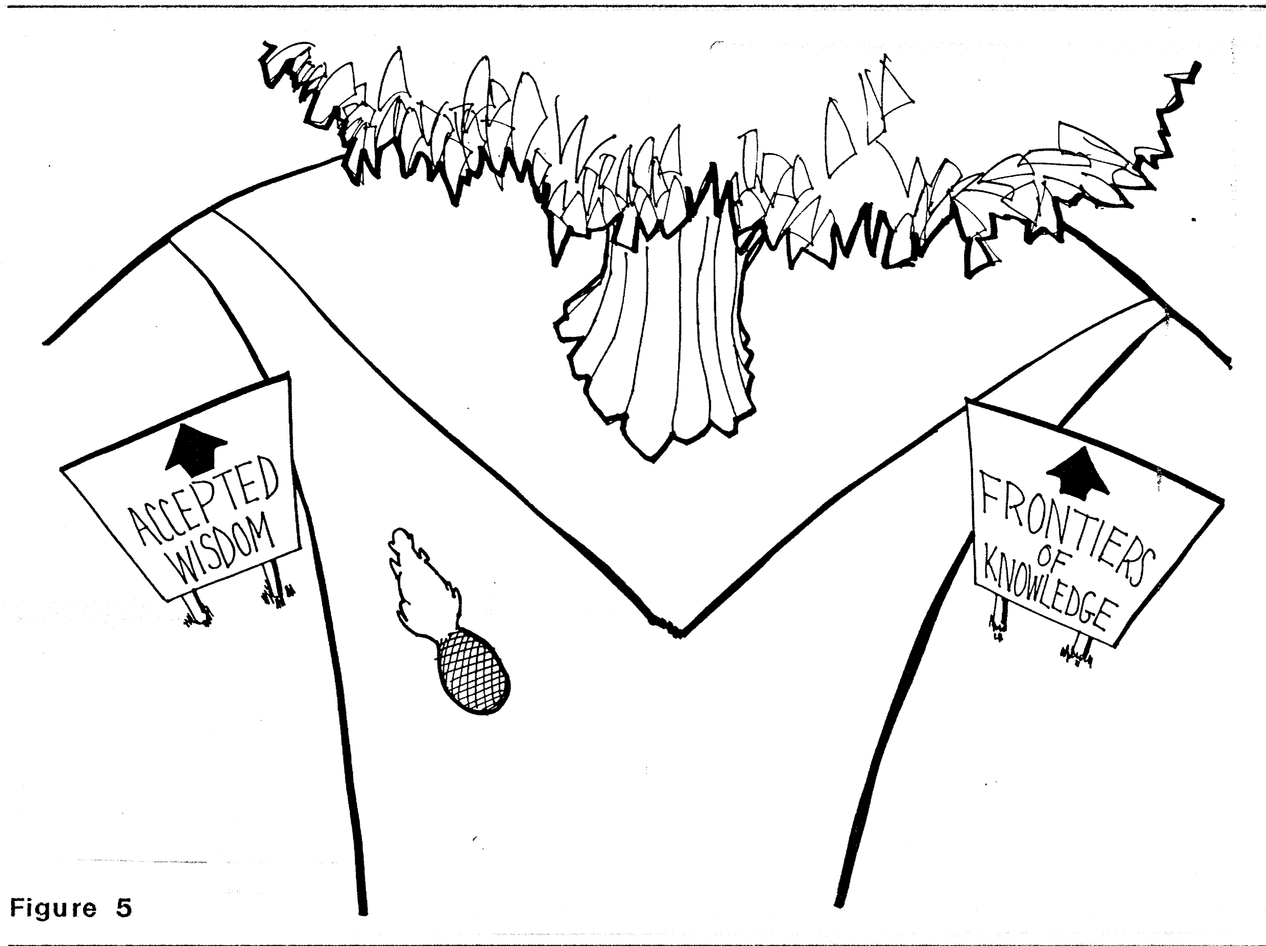


Figure 5

IN THE LAND OF ACCEPTED WISDOM

On the first stop in our journey, I'd like to consider where we are by characterizing the stages of development of a field and then use that to assess the state of expert systems work.

The first stage of development of any field is traditionally *case studies*. Ideally, we test one single dimension of a design --- one idea on knowledge representation, or control, or system architecture. Each of these is in effect an isolated point in the design space.

As the set of such experiments grows, a collection of *architectural principles* may emerge. By examining many case studies we may begin to understand the shape and character of the design space. This allows us to make empirical observations about which parts of the space make sense for which kinds of problems. Note that these are simply empirical observations --- we don't really understand why they hold; we can only say that, with our experience this design looks like the right one to use.

Eventually, and of course it's a big eventually, we get to something worthy of being labeled a science. For our purposes, that stage is characterized by an understanding that goes beyond a set of empirical observations. There is instead an understanding of *why* a particular design is appropriate for a particular task, and perhaps a much better understanding of the character and shape of the design space.

I think it's fair to say that work on expert systems is currently somewhere between the stage of case studies and architectural principles. As is traditional for fields at that stage of development, and as is particularly appropriate for expert systems, the existing knowledge is well captured as a collection of informal rules of thumb.

Knowledge Engineering Homilies

So on our next stop in the land of Accepted Wisdom, we pause for a moment to survey the sights (Fig. 6) and see what it is that we know.

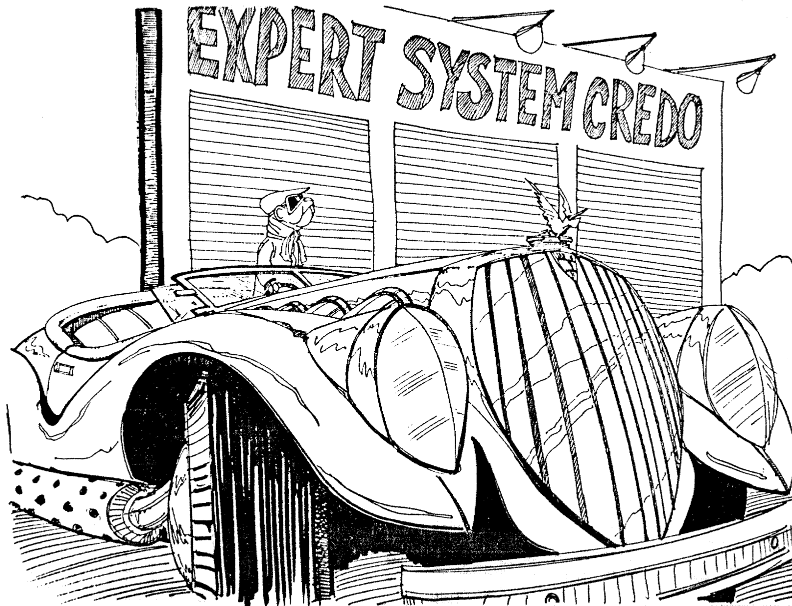


Figure 6

Figure 7 presents six of the basic commandments. Many of you will of course recognize these from a talk given four years ago by Ed Feigenbaum (Feigenbaum, 1977). I've glorified their

form, but not their content. Let me review them briefly.

CREDO

In the knowledge lies the power.
The knowledge is often inexact, incomplete.
The knowledge is often ill-specified.
Amateurs become expert incrementally.
Expert systems need to be flexible.
Expert systems need to be transparent.

Figure 7

Perhaps the most fundamental observation --- *In the knowledge lies the power* --- suggests that problem solving performance often arises from extensive stores of knowledge about the task at hand, not from a large collection of domain-independent methods.

We note that the knowledge is *inexact and incomplete*, because, almost by definition, the kinds of problems attacked in AI very rarely have complete laws or theories. Rather we have to deal with inexact and informal knowledge, the stuff of half-baked theories and guesses.

The knowledge is often *ill-specified*, because the expert can't always express exactly what it is he knows about his domain. As a result, knowledge explication becomes an important task: we need to help the expert make precise what it is he knows and how that is applied to the problem at hand.

Amateurs, whether they're people or programs, *become expert incrementally*. Unlike Athena, they don't emerge fully formed. Acquiring expertise is instead a process of incremental approach to competence.

One direct consequence of this is the importance of *flexibility* and *transparency*. Our systems are going to spend most of their lives being changed, updated, and improved. If it's too difficult to change them --- they are inflexible --- the whole process is going to come to a grinding halt. Transparency is similarly motivated: If we're going to improve the system we have to know what it did in order to determine what it is we ought to change. If the system is a black box, it becomes impossible to make that determination and system evolution will cease.

Architectural Principles

In addition to the basic credo, some architectural principles have begun to emerge (Fig. 8). One suggestion is to *separate the inference engine and the knowledge base*. By doing so, the knowledge in the knowledge base becomes more easily identified, more explicit, and more accessible. If the two are intermixed, domain knowledge will get spread out through the inference engine, and it becomes less clear what we ought to change to improve the system. The result is a less flexible system.

```
ARCHITECTURAL PRINCIPLES

Separate the inference engine and knowledge base
Use as uniform a representation as possible
Keep the inference engine simple
Exploit redundancy
```

Figure 8

A second architectural principle is *uniformity of representation*. This cuts down the number of mechanisms required, keeping system design simpler and more transparent. Each time a new representation is added to the system, something else in the system has to be able to handle it, has to know its syntax or semantics to be able to use it. Hence fewer representations means a simpler, more transparent system.

Keeping the inference engine simple helps in several ways. Explanations, for example, are easier to produce. Since they are currently generated by replaying the actions of the system, keeping those actions simple means less work is necessary to produce comprehensible explanations. Knowledge acquisition is similarly easier. When the inference engine is less complicated, less work is needed to determine exactly what knowledge to add to improve system performance.

A fourth principle -- *exploiting redundancy* -- is nicely illustrated by work on HEARSAY [Erman et al., 1980] that illustrated how redundancy can be a remedy for incomplete and inexact knowledge. The trick is to find multiple overlapping sources of knowledge with different areas of strength and different shortcomings. Properly used, the entire collection of knowledge sources can be a good deal more robust than any one of them taken alone.

The Current State of the Art

What then is the current state of the art? We can characterize it roughly as suggested in Fig. 9. Expert systems have to deal with very *narrow domains of expertise*; we have to very sharply constrain what it is we hope to achieve with them. As we get closer to the boundaries, their behavior gets rather *fragile*; rather than degrading gracefully, it tends to fall apart precipitously.

```
STATE OF THE ART

Narrow domain of expertise
Fragile behavior at the boundaries
Limited knowledge representation language
Limited I/O
Limited explanation
One expert as knowledge base "czar"
```

Figure 9

Since the emphasis and the effort in these systems is on accumulating large knowledge bases, the *knowledge representation* used is typically one of the simpler ones, like attribute-object-value triples, production rules, and so forth. And since the natural language problem

has yet to be solved, we're stuck with limited interaction languages, usually keyword-based parsing of input and template-generated production of text on the output.

Our model of explanation is useful, but limited at the moment to recapitulation of the system's actions. In many cases that's sufficiently informative that it explains system behavior.. And finally, we don't yet know very much about dealing with *multiple experts*. How do you reconcile differing and perhaps conflicting views among acknowledged experts? At the moment we don't know, so we appoint one expert as knowledge base czar and attempt to build the system in his image.

Successes of the Accepted Wisdom

So much for the current state of the art. Where does that get us? In what ways has the Accepted Wisdom led to success?

One success is the existence of the credo itself. Interestingly, not very long ago those six commandments were highly controversial. Not very long ago Joel Moses was told, when working on MACSYMA, "That isn't AI. Don't know what it is, but it isn't AI." Not very long ago the comment on DENDRAL was "That's interesting chemistry, but what has it got to do with AI?" The past five or six years have seen a significant shift in emphasis: Perhaps it *isn't* cheating to build a large store of knowledge about a domain. Perhaps part of intelligence really *does* arise from the accumulation and use of lots of information about a task.

The Accepted Wisdom has also proven to be a simple but effective tool for getting started in problems involving extensive amounts of informal knowledge. It suggests extracting knowledge from experts by working through sample problems with them to find out what they know. It suggests as well a methodology for knowledge acquisition: the detailed comparison of system vs. human performance on carefully selected problems.

The Accepted Wisdom has also given us at least four systems in current use, systems that are solving some interesting and difficult problems. DENDRAL, MACSYMA, PUFF (a program for diagnosing some classes of lung disease [Kunz et al., 1978]), and R1 (system configuration for VAXes [McDermott 1980]) are no longer the focus of work in research labs, they are instead being employed by the appropriate user community.

Those are the technical successes of the Accepted Wisdom. There have also been some important sociological effects. It first became evident at last year's AAAI conference and is clearly manifested here as well: The Accepted Wisdom has succeeded in arousing considerable interest in expert systems (and AI in general) outside the academic centers. We're beginning to see more and more industrial research centers becoming interested in the technology. Why is that? I think that part of it is simply the coming of age of the field in general. Certainly much the same thing is true in natural language and robotics where commercial systems are beginning to be available.

That interest has had a number of results, perhaps the most notable being the imbalance between the tremendous amount of interest in the technology and the rather sharply limited manpower supply capable of building these systems. That in turn has led to intensive recruiting of engineers of almost every sort (Fig. 10).

Figure 10

Calibrating the Technology

For some, this growing collaboration between industry and AI is cause for concern. There are issues of manpower spread thin over a wide range of sites and concern about manpower available to train the next generation. Equally important, however, now that AI has become a focus of strong attention, are the expectations that industry has about what AI and expert systems can supply, and how quickly that promise can be fulfilled.

It is important to calibrate as closely as we can the distance between research results and commercial products. It has become important to set expectations that make clear what can in fact be expected from expert systems technology as it currently exists.

One way to calibrate is that collection of statements about the state of the art. All those catch phrases about limited domains of expertise, fragile behavior of the boundaries, etc., help to characterize the state of the art and set the appropriate expectations about what can be done with today's technology.

To push beyond the level of catch phrases, I conducted a small survey. I called a number of people who have helped build expert systems and asked them to estimate the number of man years required. The results are shown in Fig. 11.

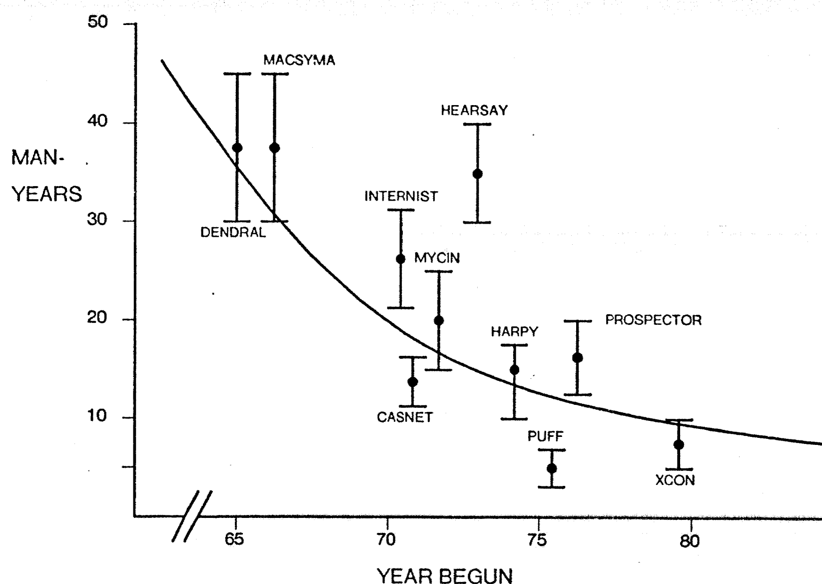


Figure 11

There are a number of interesting things here, but let me start with a few caveats about what's *not* here. For a number of reasons, the curve can't be taken too literally. First, the systems mentioned here vary drastically with respect to the size of the problem they are trying to solve. MACSYMA is probably two orders of magnitude larger than PUFF, for example. Others differ radically in the amount of knowledge they need and in the amount of effort required to build the knowledge base. So there's at least one dimension not shown --- the scale of the problem being solved.

Another dimension is the level of performance that was achieved. Some of these systems are out in the world solving problems routinely, while others are research vehicles that never made it past the laboratory stage. That too helps to acccunt for some of the difference.

But with those caveats in mind let's plunge ahead and see what the graph can tell us. Clearly there is a decline in the time required to build a system. We start off with numbers like thirty or forty man-years and come eventually to numbers only a quarter that size. I would claim in fact that the curve seems to be approaching an asymptote, somewhere around five man-years.

What accounts for the decrease? One answer is history: The older programs started first. All other things being equal, the younger programs would have accumulated fewer man-years. But this would predict only a linear reduction.

A second, more interesting answer is accumulation --- the accumulation of ideas, of code, and experience over the past fifteen years. These aren't unrelated programs --- there's a genealogy here: DENDRAL begat MYCIN, which begat PUFF; HEARSAY provided an important foundation for HARPY and then profited from its experience. We get as a result a form of the iceberg phenomenon: a great deal of work goes into the first of these in each line of succession, then the next can be built as a tip on the iceberg/foundation.

A third phenomenon that helps to account for the decline is a matching of tools to tasks. Early on --- for DENDRAL and MACSYMA --- it was the attempt to solve problems in chemistry and mathematics that led to the development of certain kinds of technology. Later on, as the technology matured, a feeling developed for appropriate applications. In a sense the positions reversed: now the tools are helping us to choose appropriate tasks. The two become as a result more closely matched, hence the shorter development time for more recent systems.

The creation of knowledge base development tools (e.g., TEIRESIAS; EMYCIN, VanMelle, 1980; or KAS, Reboh, 1980) has also accelerated the process of system construction.

But beyond all the uncertainties of the graph, one observation appears to stand out clearly. In all the experience available, the curve never drops below five man-years. I think that sets an important expectation, a very important lower bound. It implies that:

> **Even for the best-understood problems, experienced researchers using the best-understood technologies still require at least five man-years to develop a system that begins to be robust.**

That's important to keep in mind, with all its qualifiers: *the best-understood problems* --- the problem must be clear and the knowledge required to solve it well documented (rarely true); *experienced researchers* --- like most arts, this one is learned by doing, and the doing provides an important base of concepts and software that speeds subsequent efforts; *the best-understood technologies* --- the Accepted Wisdom must be well-suited to the problem. Fail to satisfy any one of these and the lower bound will climb yet higher.

One final observation about the graph. Everyone I called used the same approach to estimating the amount of time spent: They multiplied the total lifetime of the project by an estimate of the average number of full-time people involved. The interesting thing was that everyone used a number between two and five for the full-time manpower level.

Now that's not terribly profound, but it is a useful order of magnitude indicator. It means that widespread experience agrees that just a few people --- between two and five, not 20 to 50 --- function as the prime movers in building these systems. That's probably not limited to expert systems; it's likely true of large software projects in general. But what does this mean? It suggests that you can't cheat the five man-year boundary. You can't say "It's going to take five man-years, so let's get ten people and we'll be done in six months". That won't work. There are limits to what can be done with division of labor and we encounter one of those limits here.

Stages of Development

There's another way to calibrate the effort involved in building expert systems. Several years ago Ted Shortliffe and I speculated on the stages of development in the lifetime of an expert system and came up with the informal list of Fig. 12. While these may not be exactly the right stages for every project, they do offer another yardstick useful for calibrating effort.

STAGES OF DEVELOPMENT

1) System design
2) System development
3) Formal evaluation of performance
4) Formal evaluation of acceptance
5) Extended use in prototype environment
6) Development of maintenance plans
7) System release

Figure 12

The important point is that, with few exceptions, no expert system to date has made it beyond Stage 3. Fewer than a half-dozen of the many efforts have even made it even that far. Two of them --- MACSYMA and DENDRAL --- sneaked their way through as the pioneering efforts; they were the

"initialization conditions" for the field, and as such, intermixed all the phases noted here.

HEARSAY and HARPY had somewhat clearer development histories and were tested sufficiently to provide a rough calibration of accuracy on 1000-word vocabularies.

Of the systems built to date, R1 had by far the most clearly defined development process, evolving through a sequence of stages similar to those listed here. In its first formal evaluation, the system was tested on approximately 20 cases. The results suggested that R1 would soon solve problems correctly 90% of the time. This was very encouraging and indicated that the program was ready to be distributed to its user community for more extended testing. But when placed in that setting, users criticized system performance *40% of the time*.

Performance had suddenly plummeted to the 60% level. What happened? To be fair, some of those were mistakes on the part of the users resulting from incorrect data or a misunderstanding of program operation. But there was a more basic lesson: research environments, no matter how carefully tailored, are simply not identical to user environments. They differ with respect to the problem mix, how familiar users are with the program, and a range of other factors. As a result, evaluations in the research environment can be at best only rough approximations to the results expected when the program is placed in the user community.

Yet many expert systems developed in the laboratories don't even reach the stage of formal evaluation in the research environment. There's a long road between systems as they are described in conferences like this one and systems ready to be employed by a user community.

In yet another interesting observation about R1's development, McDermott notes that "R1's knowledge grew at least as much during the last stage of its development as in any of the previous four stages." The program appeared to be in its final stage of development, yet the knowledge base was still growing rapidly. What does that mean? It means that a system like this is not built and then polished up and distributed. The process is instead one of constant, incremental growth and improvement of the knowledge base. *Fundamental and important growth of the system is a process that will continue all of its useful life.* The knowledge engineering task is a continual one. The road between systems described here and systems ready for a user community is not traversed in a single jump; it's a continuing process.

One final calibration point arises from characterizing the systems described in conferences like this. Those systems are designed to test out new ideas in knowledge representation, control, acquisition, etc.; they are new sample points in the design space. They are most often Stage 2 systems; rarely have they reached Stage 3. And when the developers say the system "works," *they mean the ideas work*. Typically they have a fragile prototype which has just begun to display credible performance. That alone is generally a significant accomplishment and one with non-trivial implications: it indicates that the basic design of the system is likely to be sound. But it is a long way from being a system ready for distribution and use.

Let me summarize the calibration issue. As the curve suggests, building an expert system means a substantial investment of time and manpower. As the stages of development suggest, there is a long road between the systems described here and a system ready for a user community. As the characterization of research systems suggests, a system described here is only the beginning of a continuing development effort.

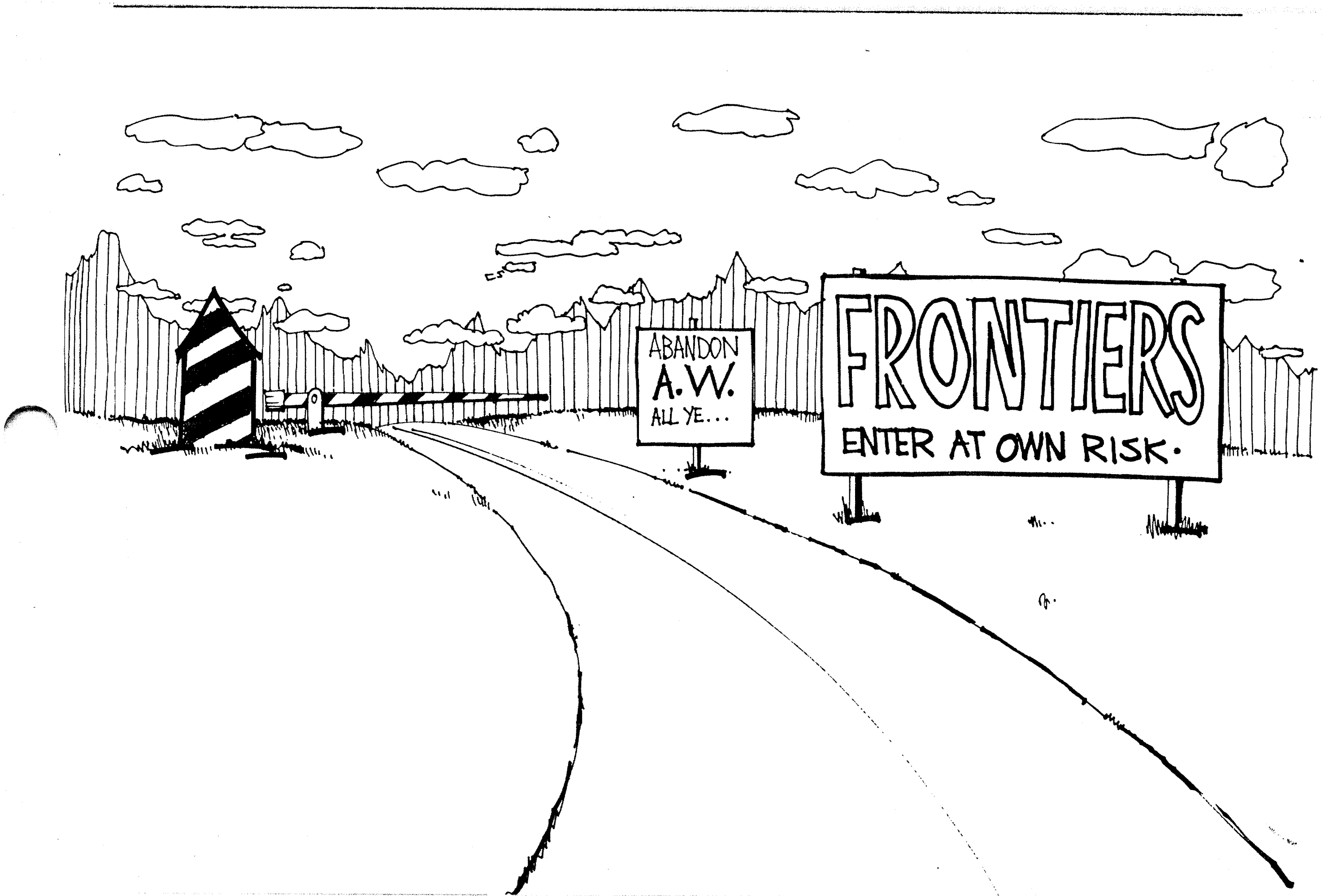


Figure 13

TOWARD THE FRONTIERS

Thus far on our journey we've seen the successes of the Accepted Wisdom, both technical and sociological, and tried to calibrate them. The successes are important and numerous, but like all such they have to be understood in the appropriate context and their power carefully measured.

Today, however, my central task is to look forward, to look for the holes in the Accepted Wisdom, the places where it starts to wear thin. To do that, let's take off toward the Frontiers. Let's see what the Accepted Wisdom can't do and find out what we have to deal with as we get closer to the Frontiers.

Where the AW Falls Short:The Architectural Principles Again

One place where the Accepted Wisdom begins to wear thin, interestingly enough, is that set of architectural principles we passed by earlier. Before we pass over that foreboding boundary at the Frontiers of Knowledge, let's take a quick glance backward at that set of principles (Fig. 14).

```
ARCHITECTURAL PRINCIPLES

Separate the inference engine and knowledge base
      grain size
      sequential information

Use as uniform a representation as possible
      Macsyma, Hearsay-II experience
      Specialization is worth the cost of translation

Keep the inference engine simple
```

Figure 14

They have not, alas, proved to be uniformly valid. Consider for example the suggestion that we should separate the inference engine and the knowledge base. There are two situations where that seems not to work out very well: (a) where the "chunks" of knowledge are inherently large, and (b) where we're dealing with basically sequential information.

Consider the issue of large grain size. The concept is not precisely defined, but let me characterize it for a rule-based representation as a rule whose "then" part has a large number of actions to carry out. That is, there are many things we want to do once we've satisfied the precondition part of the rule. How can we encode such information? Well, the obvious way is simply to list each of those actions in the order we want them carried out. But if there are a lot of them, we probably want some control over them. We may want to be careful or clever about the sequence in which they're executed. One option is to further decompose the set of actions into more rules, but there are situations where that would destroy the comprehensibility of the rule. It may hang together as one large chunk of knowledge but become lost in the trees if we chop it into a collection of rules. So we keep it all together by writing long right-hand-sides that begin to look like small programs.

But wait a moment, isn't that what the inference engine is all about? Isn't control of those actions just what the inference engine is supposed to take care of? If we build large action parts in a rule, have we really made that separation?

In domains where the knowledge falls naturally into large chunks, then, we simply may not be

able to obey the dictum that the inference engine and the knowledge base should be kept separate.

Now consider domains where the information we want to encode has a strongly sequential character. Let me return to R1 for an example. As McDermott notes, "R1's rules can be viewed as state transition operators" and, indeed, that's what they are. The whole process of configuring a system can in fact be viewed as a passage through a sequence of states.

What impact does this have on encoding knowledge? That depends on the number of such states and how recognizable they are. If there are relatively few states as compared to the number of rules, and if those states are relatively well-recognized, then it is not difficult to separate what to do (the knowledge base) from when to do it (the inference engine).

But if there are almost as many states as rules, and if the state names have no intuitive meaning and begin to look like arbitrary identifiers, well then perhaps there is a better approach. Suppose we label the states with numbers instead of names, and suppose we use a counter to keep track of which state we're in. Then, to fire the appropriate rule, all we have to do is check the number in the register and fire the corresponding rule. We could call it a "program counter."

Even though rules written that way look independent, even though they appear as individual inference steps, in fact they are locked together in a very tight and important structure --- the sequence of state transitions that defines what to do next.

Note that the point is not simply whether rules are an appropriate representation here. The point is that some information is inherently sequential. Sometimes what we want to know is that *after you've done W, do X, then do Y, then do Z.* The knowledge in such a domain is knowledge of the correct sequence of actions. In such cases the inference engine and knowledge base become nearly indistinguishable.

Our second architectural principle saw virtue in uniformity of representation. Yet there is strong evidence to the contrary: the experience of two of the larger expert systems efforts suggests exactly the opposite. The designers of MACSYMA and HEARSAY-II have both commented explicitly on how important it is to have available a range of specialized representations.

In the HEARSAY-II experience, for example, a uniform representation was useful for communication between knowledge sources, but the attempt to enforce the same uniformity on operations within a single knowledge source "... either failed completely or caused intolerable performance degradation" [Erman et al., 1980]. Instead, within a single knowledge source they used representations appropriately specialized to the task at hand. The word recognizer, for example, had its own graph network representation of words; the word sequence recognizer used a bit matrix to represent word adjacencies. The common language was used for writing on the blackboard to communicate with other knowledge sources, but within the scope of a single knowledge source the appropriate specialized representation was used.

The MACSYMA experience is similar: a multiplicity of representations makes possible the efficient implementation of a wide variety of algorithms.

The lesson here is clear: It's so expensive to work with the wrong representation that you're better off working with the right one and paying the cost. Or to put it more positively:

Specialization is often worth the cost of translation.

Our third architectural principle suggested keeping the inference engine simple. Reasonable enough, but if we take heed of the warnings we've just seen and begin to allow a range of specialized representations, we may have a problem. The inference engine has to keep step, that is, it has to be able to deal with each of those representations. And it's not clear that we can maintain simplicity if we allow the representations to multiply.

Where the AW Falls Short: Explanation and Knowledge Acquisition

One reason, then, for marching across the Frontiers is the mixed experience with the set of still-emerging architectural principles. There's another reason as well, and for that I want to go back to my comment about the nature of expertise. As we noted earlier, when the only concern is problem-solving performance, empirical associations can be a powerful technique. Collecting them may be a formidable task, but a large collection of them can be effective.

But what happens when we focus on some of the other behaviors, like explanation or knowledge acquisition? Let's consider explanation and examine the kinds of responses we get from MYCIN. If the system infers the presence of an organism called a bacteroides, and you ask why it believes that, the response is of the form, "A gram negative, anaerobic, rod-shaped organism is a typically a bacteroides." And if you ask why *that's* true... well, in fact you can't ask why that's true. I think the system ought to be able to say "Damned if I know," because in fact, damned if anybody knows. It really is simply an empirical observation that seems to hold true.

MYCIN does try to provide an answer of sorts by having justifications for rules. These are usually pointers into the literature where one can discover in medical jargon that "the ultimate etiology yet to be established" (i.e., damned if they know either).

The point is simple: the use of empirical associations precludes any more substantive form of explanation. If all the system knows is that "A and B suggest C," then it *can't* explain why, beyond repeating the rule. For domains where such rules are all that's currently known, that is indeed the best we can do. But where more fundamental insight is available we should be able to answer more substantively, and we will need representations other than simple associations to make that possible.

Next let's consider knowledge acquisition. I think there was a very interesting lesson learned several years ago in work on Meta-DENDRAL [Buchanan and Mitchell, 1978a]. The task was to induce new rules about mass spectrum analysis from examples of spectra and their analyses. One of the clear lessons from that work was that some model for understanding events in the domain was crucial for guiding and grounding the induction process. The model needn't be elaborate; Meta-DENDRAL needed little more than the ball and stick model of chemistry. Yet that model offered a simple way of understanding *why* certain molecules fragmented in certain ways, and, equally important, why some other fragmentations were highly unlikely. In doing so it provided a crucial filter capable of distinguishing plausibly meaningful hypotheses from those that were likely to be mathematical coincidences in the data. Without such a filter, the number of hypotheses generated is simply much too large; the information cannot be distinguished from the coincidental noise.

Once again, pure associations fall short. If we wish to be able to learn about the domain, we need some understanding that pushes beyond that simple form of knowledge.

ACROSS THE FRONTIER

We now know a little about what empirical associations can do for us. In some domains they can provide an interesting and useful level of performance, offer a reasonable level of explanation, and support a primitive form of knowledge acquisition. As we have seen, they begin to display important weaknesses in dealing with the last two of these, and offer virtually no support for the remaining varieties of expert behavior noted earlier. But in domains where expertise really is simply a large collection of such associations, this is likely to be the best we can do.

The question then is, what can we do in domains where more is available? In particular, when we can get mechanistic or causal models, how can we represent and use them?

Work on such questions is proceeding in several places. At MIT for example, a group of us is working on the problem of computer diagnosis in the literal sense --- detecting and identifying malfunctions in computer hardware. A group at Stanford is working on a similar problem of hardware diagnosis, while another group at MIT is developing a similar approach to medical diagnosis. In all these cases, the effort is based on representing the structure, function, and design of a device, and using that knowledge to guide the diagnostic process, whether the "device" is a RAM chip, a communications multiplexer, or a kidney.[2]

Let me illustrate this by considering our domain of digital hardware. What kinds of knowledge are required to be a good troubleshooter? One variety is knowledge about structure --- the "anatomy" of the device. We encounter it in forms ranging from block diagrams of an entire machine down to schematics indicating the devices and wires making up a single register. Information about function --- the "physiology" --- resides in such things as understanding the propagation of signals along wires, the operation of a RAM cell or multiplexer, or the function of an ALU. Knowledge of design comes from an understanding of the intent of the designer, the purpose and intended function of each component.

Because the machine is a human-designed artifact we have, at least in principle, relatively easy access to such information. Such is not often the case elsewhere: in medicine, for example, our knowledge of human anatomy and physiology is still growing, and Lord knows we can't get our hands on the design specs.

AN EXAMPLE

I'd like to use the rest of this talk to explore in some detail the kinds of technical advances be necessary to construct the next generation of expert systems. To do that we'll walk through a sample problem from the domain of field engineering, paying careful attention to the varieties of knowledge needed.

Let me preview the issues we'll encounter. First, I want to use this example to indicate how the existing technology falls short of providing the machinery we will need. We'll see in several places that a large collection of rules is simply not useful or convenient enough to be a feasible approach.

Second, we'll see the need for a range of specialized representations. There are too many different kinds of knowledge required to make feasible a uniform representation scheme.

Third, we'll need multiple representations. We need the electrical configuration so that we can reason about electrical connectivity and current flow. We need the physical configuration in order to reason about where something is or how it's connected physically to something else. And we need the functional organization, since as we'll see, we reason in large measure by observing

2. All of these efforts have benefited substantially by the foundation provided in work like that of deKleer [deKleer 1979], Sussman [Sussman 1978] and Reiger [Reiger and Grinberg 1978]. Their ideas about representing and using knowledge about structure, function, and design have provided an important foundation for thinking about causality, qualitative reasoning, and related issues.

behavior and making inferences about function.

Finally, we'll see a very simple but instructive example of what can be done when we have available something more fundamental than empirical associations. We'll see that even some very simple mechanistic models can offer significant power.

There are two caveats to keep in mind as we proceed. First, our initial efforts are focused on non-transient hardware errors. That is, we do not attack the problem of distinguishing between hardware and software errors, and will not be dealing with transients. Eventually both of these will have to be considered, but we focus first on the more tractable cases.

Second, this example indicates our ultimate goal, not our current capabilities. We have only recently begun building a system capable of the kind of representation and reasoning we're about to examine. The example illustrates the state of our aspirations, not the state of our art. The example is, however, quite real and the details are accurate.

The Problem

The phone rings at the local field service center...

"Field Service. Can I help you?"

"Yeah, my machine doesn't work!" says a frantic voice.

"What seems to be wrong?"

"Well, when I sit at the console terminal and load the system, everything's just fine. The hardware powers up, the operating system loads, and it's clear that the operating system is running. From the console terminal everything is just fine. But the user terminals don't work. They're turned on and they look OK, but when you type something, absolutely nothing happens. So what's wrong?"

Think about that for a minute. What *is* wrong? More precisely, *What kinds of information do we need in order to determine the answer to the question*?

Clearly, in order even to get started, we need some information about the *structure of the system*. Figure 15 presents the configuration of the system we're talking about. There are two main busses with an interface between them; a disk, the cpu, and memory hang off bus 1, with the user terminals on the other.

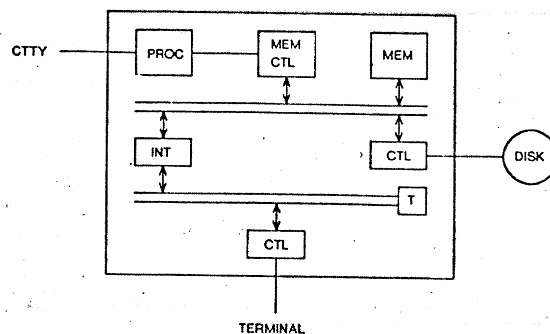


Figure 15

What does this give us? It supplies the anatomy of the system at the coarsest level of detail. And what can we do with it? Well, not too much until we supply some information about the *function* of the various components. For example, disks and memory are used to store information, busses transmit information, and cpu's execute instructions.

Now we can begin to use that information about structure and function to reason about the *behavior* of the system. Loading the system means reading the code off the disk into memory, then

executing that code. Given the structure indicated by the diagram, we know that that process must involve the disk, its controller, bus 1, the memory, memory controller, and cpu.

We can annotate our "anatomical diagram" to indicate our conclusions about how well the components are working. For example, the system load succeeded and that tells us quite a lot. It means the transmission of information from disk to memory must have succeeded, so we believe the source (the disk), the sink (memory), and the transmission medium (bus 1) are all functioning properly. The system runs, meaning that the hardware is capable of executing a program (the system code), hence the the path from memory to cpu must be functioning. Thus the memory, bus, memory controller, and memory control to cpu link are all probably functional. Finally, since the program in question --- the system itself --- is a large program with a lot of nontrivial code, we suspect that the cpu itself is working properly. Since the system clearly responds to the console terminal, we suspect that is working properly as well.

And what of the problem with the user terminals? Once again, *an understanding of behavior derives from reasoning about structure and function.* The terminals work in full-duplex mode, meaning that they can be viewed in the simplest terms as a transmitter and receiver, sending characters to the cpu and printing the characters echoed back. If they don't work then there must be something wrong with the source (the terminals), the sink (the cpu), or the transmission path. On those grounds alone we ought to suspect the terminals themselves, their controller, bus 2, the bus interface, bus 1, memory control, and cpu. But we already have pretty good evidence that the last three of those (bus 1, memory control and cpu) are all functional, so we contain our suspicions to the first four. The results of this reasoning process are summarized on Fig. 16.

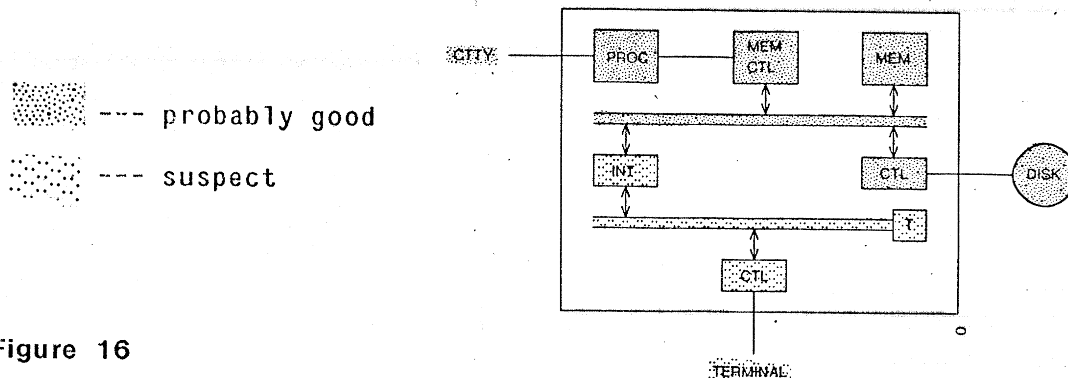


Figure 16

We can begin by narrowing our focus. All of the user terminals are malfunctioning identically (no response). We have either an extraordinary coincidence, or more likely, the problem lies "upstream," at a point in the data path common to all the terminals. We can thus direct our suspicions to the bus interface, the second bus, and the terminal controller.

Conveniently enough, there is a set of diagnostic programs designed to test much of this hardware. Life isn't always so easy (as we'll see below), but it is useful to be able to take advantage of diagnostics where they already exist.

So we retrieve the relevant set of diagnostics and run them. The results are shown in Fig. 17 (tests 1-5 gave no error messages, diagnostics were halted after test 8). What do they tell us? Once again the relevant issue is really "What kinds of information do we need in order to determine the answer to the question?"

DIAGNOSTIC RESULTS

TEST 6				
EXPECTED	AAAA	3333	F0F0	FFFF
RECEIVED	AAA8	3331	F0F0	FFFD
TEST 7				
EXPECTED	0000	0000		
RECEIVED	0031	000D		
TEST 8				
EXPECTED	0A0A	0312		
RECEIVED	0A08	0310		

Figure 17

We need to know at least two things: (a) the *intent* of the test that encountered the error, and (b) the *behavior* of the device responsible for the error. The first of these is important because the error messages make no sense unless we know what the diagnostic was trying to do. The second is important in allowing us to determine how the device under test may have failed.

If we're lucky, the intent of the test is made clear in the documentation; if we're not so lucky we may have to read the code. Clearly, no program we build anytime soon is likely to be able to do either of these. What reasonable substitute is there?

A reasonable substitute might be constructed from a suitably annotated, simplified view of the hardware. Since the author of a diagnostic is typically trying to test just a single part of the machine, much of the remainder becomes temporarily irrelevant. We can take advantage of this to provide a much simplified view of the system. Consider Fig. 18, for example. We've ignored the system disk, terminals and terminal controller, and added a level of detail to the terminator, which for our current purposes can be viewed as three registers.

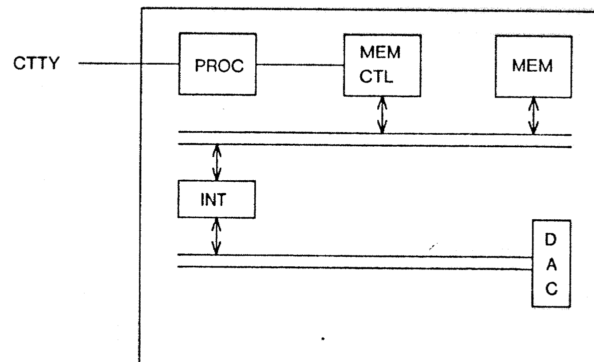


Figure 18

We then describe the diagnostics in terms of operations performed on this simplified machine: The tests in question attempt to send bit patterns back and forth between the processor and each of the terminator registers.

Now that we know what the test is trying to do, we still need to know something about the behavior of the device being tested --- a register. For our purposes a register can be viewed as a

single memory cell (Fig. 19). To make sense of the error messages, we need to understand the behavior of that device. In particular, we need to determine how its behavior changes as a consequence of various forms of failure.

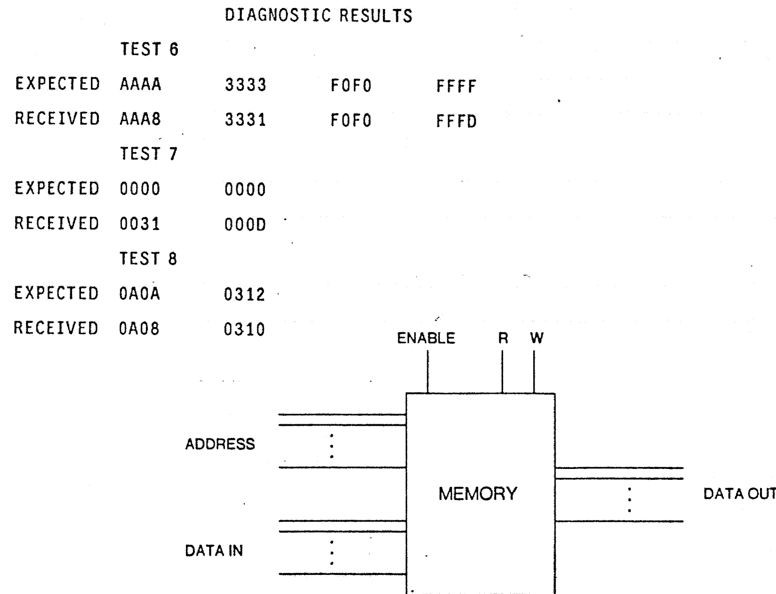


Figure 19

To do that, we need a reasonably detailed model of how it works, a model capable of supporting the sort of reasoning that says "if the write-enable line were stuck low, then no new information would ever get into the cell, and every time we read it the same thing would appear." If we could do that, then we could determine that the symptoms in Fig. 19 are not consistent with errors in any of the control lines, but *are* consistent with a dropped bit on the data-in or data-out path. We have in effect a simple induction problem, but we can *guide the induction process by using our understanding of device function to constrain the set of plausible hypotheses*.

We can thus make sense of the error messages by relying on both an understanding of what the diagnostic is trying to do and a model of the behavior of the device being tested. The result, of course, is the hypothesis that Bit #1 is being dropped somewhere on the path from the cpu to the data register (the register under test at the moment). Notice that the hypothesis is consistent with all the results of this test, both the failures (where expected and received differ) and the successes (expected and received are the same).

What can we make of Test 7? For the moment, nothing. Test 8 is once again consistent with our hypothesis. Can we come up with an hypothesis that accounts for all the data? Yes, if we invoke one reasonable heuristic: each diagnostic in a sequence is strongly dependent on the success of the previous tests. Like the syntax checks in a compiler, if one test fails, the results of successive ones can be misleading or nonsensical. So our overall guess is that we are dropping Bit #1 and that the results of Test 7 can be chalked up to the vagaries of diagnostic interdependence.[3]

To summarize the progress thus far: the initial set of symptoms suggested a problem in the communication path from the user terminals to the cpu. The fact that the system ran from the console

3. As we'll see, this is wrong.

teletype suggested that the cpu, memory, and the bus connecting them was functional. A prepackaged set of diagnostics was used to test part of the path from user terminals to cpu; the diagnostics indicated that the bug was a dropped bit.

So we have a hard error that's been found and characterized by the diagnostics. It's often suggested that a problem that well identified is effectively solved. But let's continue on, because it will become clear that there is much non-trivial reasoning left to do.

Take another look at Fig. 16. At the moment we know only that there we're dropping a bit somewhere on the path between the cpu and the data register of the terminator on Bus #2. But there's a fair amount of hardware along that path, so we might try signal tracing to find the source of the problem.

Before we go to the trouble of probing, there are some easy inferences we can make. The diagnostic gives the same results for all three of the registers in the terminator. As with the user terminals, either all three are broken in an identical fashion, or the problem lies further back along the data path. Occam's razor once again suggests the latter.

From that we get the view of Fig. 20, with the region of suspected hardware narrowed down a little. We no longer suspect the terminator and we've focused in on one bit of the data path.

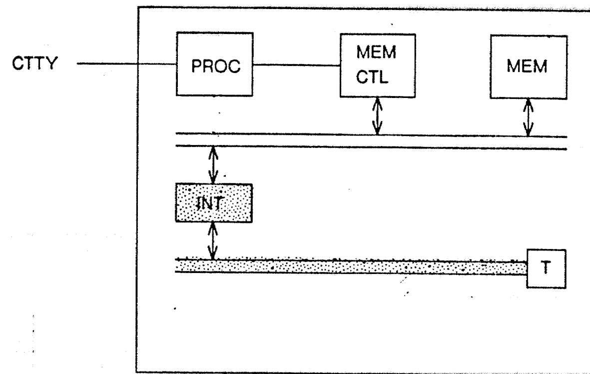


Figure 20

In keeping with that narrower focus, we need a more precisely focused test. The diagnostics are effective as broad tests of large parts of the machine, but now we need a finer probe capable of testing just that part of the machine we suspect is malfunctioning.

We want to test one bit of the datapath, but there are really two routes --- one heading out to the terminator and one heading back. They overlap substantially, but differ in places. We can test the outbound path by writing to the terminator, the inbound path by reading from it. What shall we do? The choice seems clear --- we ought to write from the cpu to the terminator --- but the reasoning is slightly subtle. We want to put a 1 on Bit #1 of the datapath. Since we never successfully wrote and read back a 1 along that bit, we don't know what's in the terminator register, hence reading from it is not a predictable test.

We might capture some of this reasoning in a general principle like "when faced with a choice use hardware known to be functional." But a more satisfying answer would tell us how to generate appropriate tests based on the sort of reasoning about structure and function at issue here.

Having decided to test the outbound path, we put the cpu in a loop that continually writes a 2 to the data register and proceed to trace the signal. Fig. 21 presents an expanded view of the path, running from the point where Bus #1 meets the bus interface, through the interface, along the backplane, and out to the data register. We've moved down one level of detail and have focused on one bit slice, but we're still dealing with an abstracted view of the hardware.

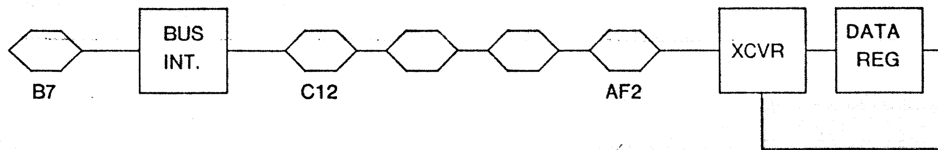


Figure 21

The hexagons are backplane pins; they're notable primarily because they're easily accessible test points That's a standard piece of diagnostic reasoning of course --- use the easy, "non-invasive" tests first, since the more complex tests often have the risk of serious side effects.

Just to make sure we're on the right track, we probe first at B7, and find the 1 is indeed making it this far (more evidence that we were correct in assuming that the cpu, memory, and Bus # 1 are functional). We probe AF2 and find that the bit is missing; probing C12 shows the same thing. We've managed to narrow the focus still further --- the signal is getting lost somewhere between Bus # 1 (pin B7) and Bus # 2 (pin C12).

Comparison with the Accepted Wisdom

One of the central points of this talk is that the Accepted Wisdom would have a difficult time supporting some of the reasoning we need to do. Let's pause for a moment and explore that point.

Consider the signal tracing we've just done and imagine trying to do it with the Accepted Wisdom. One plausible way would be to write a set of If-Then rules, that might say something like:

```
If
     the signal is OK at B7 and
     the signal is blocked at AF2
then
     the signal is being lost somewhere between B7 and AF2.


If
     the signal is OK at B7 and
     the signal is blocked at C12
then
     the signal is being lost somewhere between B7 and C12.
```

This would work and it certainly captures the effect of the signal tracing logic we used earlier, so we can hardly argue that the Accepted Wisdom is incapable of such reasoning. But it does have a number of problems. In order to do signal tracing, we need to know about the structure of the hardware. The rules above contain such knowledge, but it's bound up with the art of signal tracing and therein lies the source of many problems.

In particular, the knowledge about structure is represented in the rules only *implicitly*, it is *inaccessible* for all purposes other than signal tracing, it is *hard-wired* into the system and inflexible as a result, and it fails to recognize signal tracing as a distinct skill.

To see that the knowledge of structure is implicit, consider the rules above when viewed in

isolation. The rationale behind them is completely mysterious until we refer to Fig. 21.

To see that the knowledge about structure is inaccessible, consider trying to use it to do simulation. We ought to be able to predict the path of a signal inserted at B7, yet we can't do that without "unraveling" the information about structure embedded in the rules.

To see that the knowledge is inflexible, consider using exactly the same signal tracing technique on another piece of hardware. We would have to write out a new set of rules; none of the existing rules would be useful. Yet the underlying skill would be entirely the same.

Finally, the rules miss the sense of what signal tracing is all about. It's a process of examining structure, but it is distinct from that structure in much the same sense that an interpreter stands apart from the program it interprets.[4]

A practitioner of the Accepted Wisdom might claim that we have portrayed the approach unfairly and there is a better way to use it. Presumably this would involve writing a set of rules that tried to capture just the signal tracing skill and employing a separate description of structure and function. Yes, that's just the point. If we go to the trouble of building a separate representation of structure and function, we ought to do it right, which means a hierarchy of multiple levels of detail, information about connectivity, substructure, a language for behavior, etc.

The point is simply that the Accepted Wisdom *focuses on the use of rules embodying empirical associations.* It does not offer us any tools for constructing descriptions of the sort we need, it does not offer us any techniques for using those descriptions to guide diagnosis, and perhaps even more important, *it does not even lead us to think in such terms.*

The Example Continued

Let's return to the example. We've determined that the problem lies somewhere between B7 and C12, so let's look at that in more detail (Fig. 22). Bit #1 comes in at B7 on the left, passes through the gate array, into the transceiver in chip E4, and out again to pin C12. We're losing it somewhere between those two points.

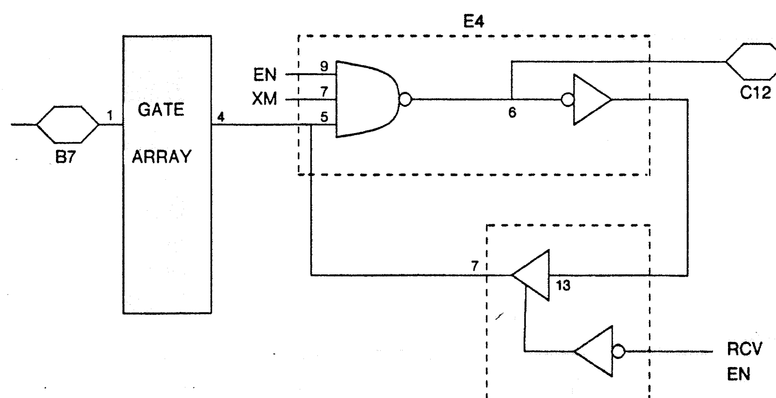


Figure 22

How can we locate the source of the problem? There's more signal tracing to be done, but once again let's try to be as clever as we can before we go probing.

Experience tells us that gate arrays have a long time to failure, so we expect that the signal

4. These failings are not unique to the problem of structure representation. Many of the same criticisms apply in trying to use rules to encode knowledge about a sequence of timing events, like a communication protocol (e.g., see [Bennett 1981]).

will make it through to the transceiver. Let's also assume that, unless we have evidence otherwise, wires are functional; if a signal appears at one end, then it will appear at the other end.

If the signal makes it to the transceiver, what might prevent it from emerging? If the ENable or XMit signals on the nand gate were missing, then the bit would be blocked. But it turns out that both the ENable and XMit signals are shared across four bits of this data path. Hence if either one of them were bad, we would have noticed four bits dropped, not simply one.

Note the hypothetical reasoning we used. We postulated a fault and then asked whether that was consistent with the symptoms noted. That's a useful and common enough form of reasoning, but is more subtle than it might appear. We can't, for instance, simply insert a fault and then ask "How does the machine behave?" There's too much hardware there and too wide a variety of behavior to ask such a broad question. We need instead a very directed and constrained form of simulation that asks how a fault would effect the particular piece of behavior under study. In this case that tells us that the ENable and XMit lines are likely to be functional.

So it seems that the signal probably made it from B7 through the gate array and through the nand gate of the transceiver. By checking pin 6 of chip E4 we can easily verify all of this, and indeed the signal is good there.

Well then, what's going on? We said earlier that we will assume wires are functional and propagate signals. From the diagram we have nothing more than a wire between chip E4 pin 6 and backplane pin C12, so we had better check our assumption about it being good. To do that, we need to *shift representations*, and consider the *physical* structure underlying the path between pin 6 and pin C12. That structure is suggested in Fig. 23 --- there is an etch on the board than runs from pin 6 out to the pad at the edge. We can easily check that with an ohmmeter and it proves to be good.

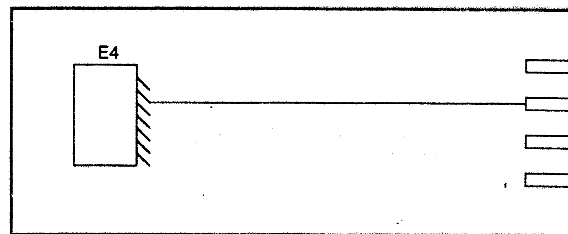


Figure 23

Going back to Fig. 22, then, we seem to have a bit of a contradiction on our hands. We have the signal getting to C12 on the diagram and yet it doesn't get to the "real" C12 pin on the backplane. To unravel the problem we once again have to shift from the predominantly electrical view of Fig. 22 to a physical view, Fig. 24. That figure makes it clear that the board fits into a slot of the backplane. The "physics" of that arrangement makes it clear that between the board and the backplane pin there is a set of spring-loaded connectors. And in this case one of them is bent and not making contact. There's our dropped bit.[5]

---

5. In truth, it was actually a piece of tape that had been inserted to simulate the fault. But these connectors do, in fact, get broken or bent out of shape by people who are a little too rough in removing and reinserting boards (as we occasionally have been).
But what about Test 7, whose error message didn't fit into the dropped bit hypothesis? As it turns out, there is a bug in that diagnostic (someone wrote a multiply where they meant a left-shift). This is yet another reason why it's important to be able to interpret test results in context, not just accept them.
With the bug fixed, results of Test 7 are consistent with the others.

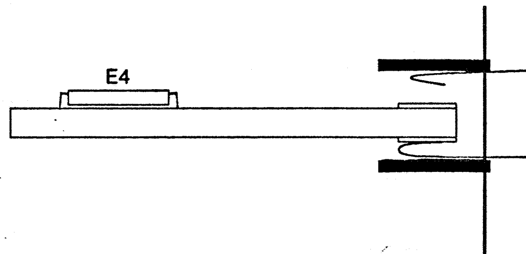


Figure 24

The Example: Summary

The example, even as simple as it is, helps to illustrate a number of points about this undertaking (Fig. 25). First, the central thrust of this work is to do diagnosis based on an understanding of structure and function. From the outset, knowledge of function (loading the operating system, typing on a terminal) and structure (system configuration) interacted to allow us to infer the likely health of various components of the system. We were able to focus our efforts because we knew how devices were interconnected, knew how they were supposed to behave, and knew something about how they were likely to fail.

```
                ISSUES

Diagnosis based on structure, function
Utility of causal models
Need for multiple, specialized representations
        physical
        electrical
        abstract function
Shortcomings of the Accepted Wisdom
The strong focusing effect
        structure
        experience [the Accepted Wisdom]
```

Figure 25

This understanding also led to a more precise diagnosis than might have resulted from the traditional approach of module swapping. Early in the diagnosis it became clear that the bus interface was dropping a bit. Had we replaced the existing board with a new one, at best that would have failed to solve the problem. At worst, the act of removing and replacing a board might have jiggled the connector sufficiently that we got a temporary contact. The same problem might then reoccur, while in the meantime a perfectly good board was sent to be repaired.

Second, we illustrated how even very simple causal models might be used as a way of "understanding" behavior. A model of the function of a RAM cell, for example, made it possible to reason about the likely cause of malfunction.

Third, we saw how a multiplicity of specialized representations aided the reasoning process. We used several different views of the machine, including electrical (the schematics), physical (the board in its slot), and a more abstract functional view (boxes labeled "transceiver," etc.). Each in turn was useful in understanding how the system worked.

Fourth, we have illustrated how the Accepted Wisdom, with its focus on collecting large numbers of empirical associations, fails to provide some of the intellectual framework and tools required for solving this problem. In particular, it fails to supply a mechanism appropriate for describing and reasoning about structure and function.

Fifth, the example demonstrates the interesting phenomenon illustrated in Fig. 26. Consider the enormous focusing that occurred as our analysis proceeded. We went from "there's something wrong with the machine," to "there's one broken connector in the backplane that's causing one bit to be dropped." How did we manage to focus in so sharply in relatively few steps? Clearly the structure of the machine itself was in part responsible: Computers are traditionally conceived of in terms of successive levels of structural and functional organization, and we put that to good use.

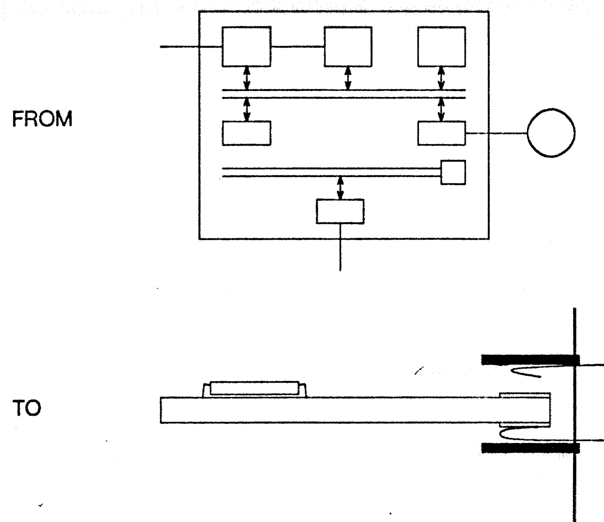


Figure 26

But part of the responsibility lies in a contribution made by the Accepted Wisdom, in the form of empirical associations and "compiled experience" about where to go looking. It's clear, for example, that a troubleshooter doesn't respond to a problem by writing down Ohm's law for the entire machine and trying to account for the malfunction in those terms. Instead what he appears to do is to ask "Where have I seen this before and what was the problem last time?" And when we had some guess about where the problem might be, we retrieved a set of diagnostics simply because we "knew" they were appropriate for the part of the system under examination. No deeper reasoning was involved, nor was it necessary. The Accepted Wisdom may thus have a role to play in this problem, functioning as a focusing mechanism. It helped us to zero in on components of the machine that were appropriate for more detailed examination.

CHALLENGES

Since this work is still in its early stages, we have only begun to construct the machinery needed to produce the required reasoning behavior [Davis 1982]. Let me comment on a few of the more important challenges we face, both in reasoning from structure and function, and in combining new directions with the Accepted Wisdom.

Diagnosis from Structure and Function

If we are to base diagnostic reasoning on knowledge of structure and function, then clearly we need some way of describing both. Structure description is fundamentally topology, the connectivity of components; a description of function must allow us to reason about how a device works and how it behaves when it fails.

Describing Structure

A number of structure description languages have been developed, but most, having originated in work on machine design, deal exclusively with *functional* components, rarely making any provision for describing *physical* organization. This is curiously true even for languages billing themselves as *computer hardware description languages.* They rarely mention a piece of physical hardware. In doing machine diagnosis, however, we are dealing with a collection of hardware whose functional and physical organizations are both important. The same gate may be both (*i*) functionally

a part of a multiplexor, which is functionally a part of a datapath, etc., and (*ii*) physically a part of chip E67, which is physically part of board 5, etc. Both of these hierarchies are relevant at different times in the diagnosis and both are included in our structure description language.

In our work we use the functional hierarchy as the primary organizing principle because our basic task involves reasoning from function (malfunction) to structure (structural defect) rather than the other way around --- we are typically confronted with a machine that misbehaves, not one that has visible structural damage. The functional organization is also typically richer than the structural (more levels to the hierarchy, more terms in the vocabulary), and hence provides a useful organizing principle for the large number of individual physical components. Compare, for example, the functional organization of a board (e.g., a memory controller with cache, address translation hardware, etc.) with the physical organization (1 pc board, 137 chips).

Since our ultimate intent is to deal with hardware on the scale of a mainframe computer, we need terms in the vocabulary capable of describing levels of organization more substantial than the terms used at the circuit level. We can, for example, refer to *horizontal*, *vertical*, and *bitslice* organizations, describing a memory, for instance, as "two rows of five 1K ram's". We use these specifications in two ways: as a description of the organization of the device and a specification for the pattern of interconnections among the components.

Our eventual aim is to provide an integrated set of descriptions that span the levels of hardware organization ranging from interconnection of individual modules, through higher level of organization of modules, and eventually on up through the register transfer and PMS level [Bell 1971].

The structural description of a module is expressed as a set of commands for building it. That is, a device is described by a set of commands indicating what subparts are needed and how those subparts are to be interconnected. These commands are then executed by the system, causing it to build data structures that model all the components and connections. The resulting data structures are organized around the individual components.

This approach offers a useful merging of the procedural and object-oriented styles of representation. We describe a device by indicating how to build it (the procedural view), but then want to think about it as a collection of individual objects (the object-oriented view). The first view is convenient for *describing* structure, the second makes it easy to *answer questions* about it, questions like connectivity, location, etc., that are important in signal tracing and other troubleshooting techniques. The two descriptions are unified because the system simply "runs" the procedural description to produce the data structures modeling the device. This gives us the benefit of both approaches with no additional effort and no chance that the two will get out of sync.

Describing Function

Behavior description is a more difficult undertaking. Several different approaches have been tried, ranging from state transition diagrams, to petri nets, to rules encoding input/output specifications, and chunks of code. Simple rules are useful where device behavior is uncomplicated, petri nets are useful where the focus is on modeling parallel events, and unrestricted code is often the last resort when more structured forms of expression prove too limited or awkward. Various combinations of these three have also been explored.

Our initial implementation uses constraints [Sussman 1980] to represent behavior. Conceptually, a constraint is a bundle of rules indicating the relationships among the values at the terminals of a device. A set of constraints is a relatively simple mechanism for specifying behavior, in that it offers no obvious support for expressing behavior that falls outside the "relation between terminals" view. The approach also has known limits. For example, constraints work well when dealing with simple quantities like numbers or logic levels, but run into difficulties if it becomes necessary to work with symbolic expressions.[6]

This approach has, nevertheless, provided a good starting point for our work and offers

some useful advantages. For example, by keeping track of dependency information --- an indication of how the system determined the value at a terminal --- we can trace backward to the source of misbehavior.

Our use of a hierarchic approach also makes multi-level simulation very easy. In simulating any module we can either run the constraint associated with the terminals of that module (simulating the module in a single step), or "run the substructure" of that module, simulating the device according to its next level of structure.

We believe it is important in this undertaking to include descriptions of both design and implementation, and to distinguish carefully between them. A wire, for example, is a device whose behavior is specified simply as the guarantee that a logic level imposed on one of its terminals will be propagated to the other terminal. Our structure description allows us to indicate the *intended* direction of information flow along a wire, but our simulation is not misled by this. This is, of course, important in troubleshooting, since some of the more difficult faults to locate are those that cause devices to behave not as we know they "should", but as they are in fact electrically capable of doing. Our representation machinery allows us to separate design specifications from implementation.

Troubleshooting

The traditional approach to troubleshooting digital circuitry (e.g., [Breuer 1976]) has, for our purposes, a number of significant drawbacks. Perhaps most important, it is a theory of *test generation*, not a theory of *diagnosis*. Given a specified fault, it is capable of determining a set of input values that will detect the fault (ie, a set of values for which the output of the faulted circuit differs from the output of a good circuit). The theory tells us how to move from faults to sets of inputs; it provides little help in determining what fault to consider, or which component to suspect.

These questions are a central issue in our work for several reasons. First, the level of complexity we want to deal with precludes the use of diagnosis trees, which can require exhaustive consideration of possible faults. Second, our basic task is repair, rather than initial testing. Hence the problem confronting us is "Given the following piece of misbehavior, determine the fault." We are not asking whether a machine is free of faults, we know that it fails and know how it fails. Given the complexity of the device, it is important to be able to use this information as a focus for further exploration.

A second drawback of the existing theory is its use of a set of explicitly enumerated faults. Since the theory is based on boolean logic, it is strongly oriented toward faults whose behavior can be modeled as some form of permanent binary value, typically the result of stuck-ats and opens. One consequence of this is the paucity of useful results concerning bridging faults.

A response to these problems has been the use of what we may call the "violated expectation" approach [Brown 1981]. The basic insight of the technique is the substitution of violated expectations for specific fault models. That is, instead of postulating a possible fault and exploring its consequences, the technique simply looks for mismatches between the values it expected from correct operation and those actually obtained. This allows detection of a wide range of faults because misbehavior is now simply defined as anything that isn't correct, rather than only those things produced by a struck-at on a line.

This approach has a number of advantages. It is, first of all, fundamentally a diagnostic technique, since it allows systematic isolation of the possibly faulty devices, and does so without having to precompute fault dictionaries, diagnosis trees, or the like. Second, it appears to make it

---

6. What, for example, do we do if we know that the output of an or-gate is 1 but we don't know the value at either input? We can refrain from making any conclusion about the inputs, which makes the rules easy to write but misses some information. Or we can write a rule which express the value on one input in terms of the value on the other input. This captures the information but produces problems when trying to use the resulting symbolic expression elsewhere.

unnecessary to to specify a set of expected faults (we comment further on this below). As a result, it can detect a much wider range of faults, including any systematic misbehavior exhibited by a single component. The approach also allows natural use of hierarchical descriptions, a marked advantage for dealing with complex structures.

This approach is a good starting point, but has a number of important limitations built into it. Most fundamentally, as we demonstrate in [Davis 1982], the traditional use of it fails to distinguish between design and implementation. Without knowing both how the device is supposed to work and how it is physically capable of working, we cannot deal with a number of common varieties of fault, like bridges and power losses.

Suggestions about repairing this problem lead to a larger issue. In working on a difficult diagnostic problem, human engineers often make a number of unjustified assumptions that serve simply to make the problem tractible. Yet they are not misled if some of those assumptions turn out to be false. We seek to create a system that operates in a similar spirit --- it should be capable of making assumptions to keep the problem tractible, yet be able to discard each assumption, determine the consequences, and continue toward a solution.

More General Issues

In addition to the specifics of troubleshooting via knowledge about structure and function, a number of more general challenges emerge from the suggestions made earlier.

It's easy to suggest, for example, that we'll have to use multiple, specialized representations. It's somewhat harder to determine the appropriate representation for each and to know which one to use when. We used a simple strategy in the problem above, starting with the functional organization, using the electrical organization next, and employing the physical representation last. This seems to be a reasonable default, since it reflects, to first order, at least, the levels of abstraction in the design of the machine. Functional organization provides the framework within which an electronic implementation is designed, and the electronics provide a framework for the physical organization. But this is at best a first order approximation. Life is rarely that well organized in design; we have no reason to expect it will be that easy in diagnosis either.

One of the longer term challenges we face is reconciling increased complexity of these systems with the desire, well-articulated by the Accepted Wisdom, of maintaining transparency and flexibility. Structural and functional models may increase the power and depth of understanding displayed by the system, but they will inevitably increase the complexity as well. How then can such systems continue to be transparent to the user and easily modified? Without transparency, they are unlikely to be widely accepted and our efforts may be in vain. Without the flexibility, they are likely to ossify long before they are fully developed, and we run the risk of never being able to finish.

One of the convenient bits of handwaving I did in the example was to dip occasionally into reasoning via empirical associations, rather than the reasoning from structure and function on which most of the example was based. As noted, both can be useful, but the obvious question is, how are we to determine which to use when? The empirical associations can supply a quick and crude guide about where to go looking, but how do we know when to shift over to more detailed models?

Finally, there is an interesting and difficult problem of selecting the appropriate level of abstraction. In the example above I selected just the right level for each step of the reasoning, providing a powerful way of simplifying the reasoning. Consider, for example, the enormous amount of detail that is suppressed in our initial view of the system (Fig. 15). But it isn't obvious how this selection process could be automated. How do we know what level of detail is sufficiently simple that it makes reasoning easy, yet sufficiently detailed that it isn't misleading?

To illustrate just how challenging this problem can be, let me show you another example. To do that I'm going to shift domains for a moment and take a problem from design rather than diagnosis, but I think it illustrates the issue nicely.

Figure 27 shows a VLSI design for a single bit of random access memory. To see how this works, consider the simplified diagram on the right. To store a bit, put the appropriate logic level on the wire at the top and close the switch labeled "W" ("write"). If the logic level is high, that will close the switch at the bottom labeled "b"; if the logic level is low, that will open the switch. The bit is thus stored as the state of the switch at the bottom.

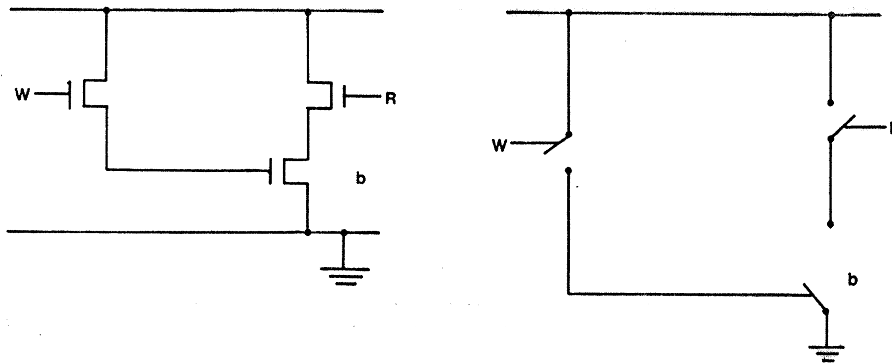


Figure 27 -- VLSI Memory Cell

To read the information out, we have to charge up the wire at the top and then close the "read" switch (labeled "R" in Figure 24). If the charge drains away to ground then the switch at the bottom must be closed, hence we must have stored a 1 there. If the charge stays around, then the switch must be open, so we must have written a 0 there.

The design sounds perfectly plausible at that level of description, but will it work? Transistors are considerably more sophisticated than the simple switch we've been using to model then. A number of more complex models are available, so one approach would be to try simulating the design with successively more complex models. We can model it as a switch with some inherent resistance, we can add capacitances, we can view it as a voltage-controlled current source, and all of these more complex models will still suggest that the design is sound.

But the device doesn't work; the design is indeed faulty. It doesn't work because the transistor, in addition to all the other things it is, is also a leaky switch. It never quite turns off completely.

When we write a 1, we charge up the base of transistor b, then let the write line go low, turning off the write transistor. But it doesn't turn off completely, and eventually enough charge leaks off the base of b that it is no longer "on." The bit has decayed away.

The leak of charge through the write transistor is, for almost all purposes, too small ever to worry about. Typically we deal with time scales on the order of nano- and microseconds, and on that time scale the leak doesn't amount to much. But remember, this device is a memory. And we'd prefer it retain what we told it, not just over the course of microseconds but over seconds, hours, and days. And on that time scale, the leak is big enough that it can result in loss of information.

What does all that illustrate? It shows that even in a very simple design, it may not be easy to determine what level of model to use. It isn't simply complexity that makes necessary the more elaborate models; it is instead something more like sensitivity. We have to ask what the device is most sensitive to. And in this case, it turns out to be enormously sensitive to an effect which is, in almost all other situations, so small as to be negligible.

SUMMARY

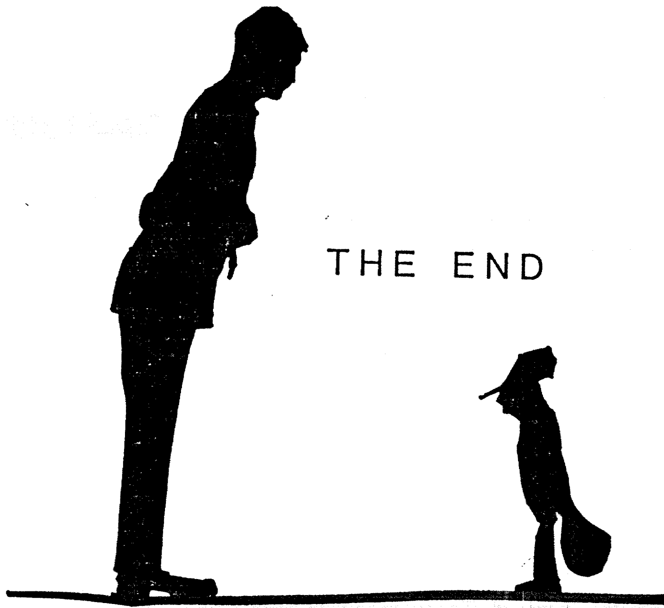
And it is here, at the Frontiers of Knowledge, that we come at last to the end of our journey. What have we seen? We started out in the Land of Accepted Wisdom, where we explored the nature of expertise, finding it to be a whole collection of interesting behaviors, of which only a very few have begun to be automated. We surveyed the extent of the Accepted Wisdom, finding it effective where expertise consists of large numbers of empirical associations, "compiled experience."

We then stepped back to evaluate the state of the art. Two important calibration points were (a) the magnitude of manpower investment necessary to build a robust expert system, and (b) the stage of development typically reached by current systems. Data from existing efforts, though meager, seems to suggest that even in the best of cases at least five man years of effort are necessary before an expert system begins to perform reliably. It is also revealing to note that most expert systems to date have been developed only through the stage of construction of the basic knowledge base. Few have proceeded to the extended testing, further development, documentation, etc., that must occur before these systems are ready for distribution to a user community.

In approaching the Frontiers, we uncovered some of the shortcomings of the Accepted Wisdom, finding that some of the accepted architectural principles may not in fact be as feasible or desirable as first expected. It proves difficult, for example, to separate the inference engine and knowledge base when knowledge in the domain is inherently sequential or where the chunks of knowledge are large. Uniformity of representation can exact a considerable price, while the efficiencies of specialization can often more than offset the cost of translating between representations.

In journeying across the Frontiers, we explored a problem from the domain of hardware troubleshooting to illustrate the kinds of challenges that we face in constructing the next generation of expert systems. We found that the Accepted Wisdom was capable of encoding some of the knowledge we needed, but discovered that using simple rules can render the underlying knowledge inaccessible, hard-wired, and inflexible. This illustrated the utility of being able to represent and reason about structure and function. We saw that even simple causal models can provide a mechanism for "understanding" behavior of devices. We used a number of different, specialized representations that included physical, electrical, and abstract functional views of the device. And finally, we saw that while all these ideas can be an effective foundation for diagnostic reasoning, there is a whole range of difficult and interesting problems to be solved before we travel very far across the Frontier.

And with that, my small friend and I will remove our official Tourguide hats, take our bows, and depart...


THE END

REFERENCES

[Bell 1971]
Bell G, Newell A, *Computer Structures: Readings and Examples*, McGraw-Hill, 1971.

[Bennett 1981]
Bennett J S, Hollander C R, DART: An expert system for computer fault diagnosis, *Proc. IJCAI-81*, pp. 843-845, 1981.

[Breuer 1976]
Breuer M, Friedman A, *Diagnosis and Reliable Design of Digital Systems*, Computer Science Press, 1976.

[Brown 1981]
Brown J S, Burton R, deKleer J, Pedagogical and knowledge engineering techniques in the SOPHIE systems, Xerox Report CIS-14, 1981.

[Buchanan 1978]
Buchanan B, Mitchell T, Model-directed learning of production rules, in *Pattern-Directed Inference Systems*, Waterman and Hayes-Roth (eds).

[Davis 1979]
Davis R, Interactive transfer of expertise, *Artificial Intelligence*, **12**:121-157, 1979.

[Davis 1982]
Davis R, et al., Diagnosis based on description of structure and function, *Proc AAAI-82*, August 1982.

[deKleer 1979]
deKleer J, Causal and teleological reasoning in circuit recognition, MIT AI-TR 529, Computer Science Department, MIT, September 1979.

[Erman 1980]
Erman L, et. al, The Hearsay-II speech understanding system: integrating knowledge to resolve uncertainty, *Computing Surveys*, **2**:213-254, June 1980.

[Feigenbaum 1977]
Feigenbaum E A, The art of artificial intelligence: I. Themes and case studies in knowledge engineering, *Proc IJCAI-77*, pp 1014-1029, 1977.

[Kunz 1978]
Kunz J C, Fallat R J, McClung D H, Osborn J J, Votteri B A, Nii H P, Aikins J S, Fagan L M, and Feigenbaum E A, A physiological rule-based system for interpreting pulmonary function test rules. Stanford Memo HPP-78-19, Stanford University (Computer Science Dept.), November, 1978

[Lindsay 1981]
Lindsay R, Buchanan B G, Feigenbaum E A, Lederberg J, Applications of AI For Organic Chemistry: the DENDRAL Project, McGraw-Hill.

[Macsyma 1974]
Macsyma group, The Macsyma Reference Manual, MIT, 1974.

[McDermott 1980]
McDermott J, R1: The formative years, *AI Magazine*, **2**:21-29.

[Mitchell 1978]
Mitchell T, Version spaces: an approach to concept learning, Stanford CSD-78-711, December 1978.

[Reboh 1980]
Reboh R, A knowledge acquisition environment for expert consultation systems, PhD dissertation, Dept. of Mathematics, Linkoping University, Sweden.

[Reiger 1978]
Reiger C J, Grinberg M, A system of cause-effect representation and simulation for computer-aided design, in *Artificial Inteligence and Pattern Recognition in Computer Aided Design*, Latombe, ed, pp. 299-334.

[Shortliffe 1976]
Shortliffe E H, *Computer-based medical consultations: Mycin*, American Elsevier, 1976.

[Sussman 1978]
Sussman G J, Slices: at the boundary between analysis and synthesis, in *Artificial Intelligence and Pattern Recognition in Computer Aided Design*, Latombe, ed, pp. 261-298.

[Sussman 1980]
Sussman G, Steele G, Constraints - a language for expressing almost-hierarchical descriptions, *AI Journal*, Vol 14, August 1980, pp 1-40.

[Swartout 1981]
Swartout W, Explaining and justifying expert consulting programs, *Proceedings 7th IJCAI*, Vancouver, Canada, 1981, pp. 815-822.

[VanMelle 1980]
VanMelle W J, A domain-independent producton-rule system for consultation programs, PhD dissertation, Computer Science Department, Stanford University, 1980.