MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Memo No. 1059

July 1988

Model-Based Reasoning: Troubleshooting

by

Randall Davis
Walter C. Hamscher

Abstract

To determine why something has stopped working, it's useful to know how it was supposed to work in the first place. That simple observation underlies some of the considerable interest generated in recent years on the topic of model-based reasoning, particularly its application to diagnosis and troubleshooting. This paper surveys the current state of the art, reviewing areas that are well understood and exploring areas that present challenging research topics. It views the fundamental paradigm as the interaction of prediction and observation, and explores it by examining three fundamental subproblems: Generating hypotheses by reasoning from a symptom to a collection of components whose misbehavior may plausibly have caused that symptom; testing each hypothesis to see whether it can account for all available observations of device behavior; then discriminating among the ones that survive testing. We analyze each of these independently at the knowledge level, i.e., attempting to understand what reasoning capabilities arise from the different varieties of knowledge available to the program.

We find that while a wide range of apparently diverse model-based systems have been built for diagnosis and troubleshooting, they are for the most part variations on the central theme outlined here. Their diversity lies primarily in the varying amounts and kinds of knowledge they bring to bear at each stage of the process; the underlying paradigm is fundamentally the same.

Our survey of this familiar territory leads to a second major conclusion of the paper: Diagnostic reasoning from a model is reasonably understood. Given a model of behavior and structure, we know how to use it in a variety of ways to produce a diagnosis. There is, by contrast, a rich supply of open research issues in the modeling process itself. In a sense we know how to do model-based reasoning; we don't know how to model the behavior of complex devices, how to create models, and how to select the "right" model for the task at hand.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research has been provided in part by the Digital Equipment Corporation, in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-85-K-0124, and in part by Wang Corporation.

The views and conclusions contained in this document are those of the author, and should not be interpreted as representing the policies, either expressed or implied, of the Digital Equipment Corporation or of the Department of Defense.

To appear in *Exploring Artifical Intelligence*, H.E. Shrobe, ed., Morgan Kaufmann Publishers, 1988

Contents

1 INTRODUCTION

In recent years considerable interest has been generated around the topic of model-based reasoning, particularly its application to diagnosis and troubleshooting. This paper surveys the current state of the art, reviewing areas that are well understood and exploring areas that present challenging research topics. It begins by describing the nature of the task, exploring what is given and what we're trying to produce. Since, as will become clear, there are considerable advantages to reasoning from a model of structure and behavior, we need representations for both; we review the set of techniques in current use and examine their strengths and weaknesses.

A considerable part of the paper is then devoted to how those representations are used to do model-based diagnosis. We view the fundamental paradigm as the interaction of prediction and observation, and explore it by examining its three fundamental subproblems: *Generating* hypotheses by reasoning from a symptom to a collection of components whose misbehavior may plausibly have caused that symptom; *testing* each hypothesis to see whether it can account for all available observations of device behavior; then *discriminating* among those that survive testing. In any real system these three are likely to be intertwined for reasons of efficiency. We treat them independently to simplify the presentation and because our goal is a knowledge-level analysis — an understanding of what reasoning capabilities arise from the varieties of knowledge available to the program.

The presentation is structured as a sequence of increasingly elaborate examples, starting with the simplest approach and adding successively more knowledge, producing successively more constraints that can be brought to bear. This is useful both as a way of simplifying the presentation and as a way of making another of the major points of this paper: While a wide range of apparently diverse model-based systems have been built for diagnosis and troubleshooting, they can all be seen as exploring variations on the basic paradigm outlined here. Their diversity lies primarily in the varying amounts of and kinds of knowledge they bring to bear at each stage of the process.

Our survey of this familiar territory leads to a second major conclusion of the paper: Diagnostic reasoning from a tractable model is largely well understood. That is, given a model of structure and behavior of tolerable complexity, we know how to use it in a variety of ways to produce a diagnosis. Part of the evidence for this is the number of different applications of that same paradigm in a variety of domains.

There is, by contrast, a rich supply of open research issues in the modeling process itself. While to some degree know how do model-based reasoning, we don't know how to model complex behavior, how to create models, and how to select the "right" one for the task at hand. The last major section of the paper deals with these topics, exploring the kind of difficulties that arise and using them to outline some important research problems.

2 THE BASIC TASK

The basic paradigm of model-based reasoning for diagnosis can best be understood as the interaction of observation and prediction (Figure 1). In one hand we have the actual device, typically some physical artifact whose behavior we can observe. In the other hand we have a model of that device that can make predictions about its intended behavior. Observation indicates what the device is actually doing, prediction indicates what it's supposed to do. The interesting event is any difference between these two, a difference termed a *discrepancy.*

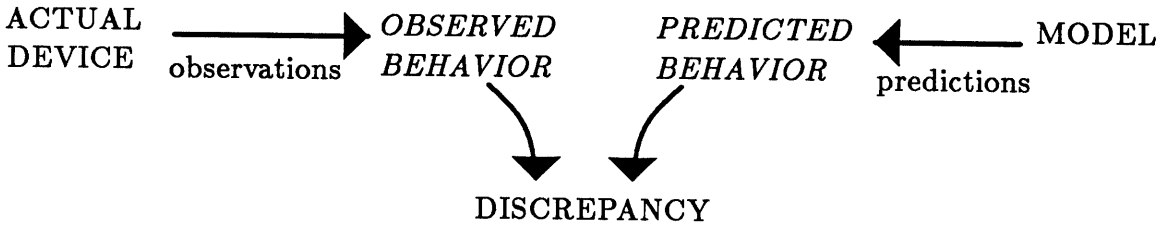


Figure 1: Diagnosis as the Interaction of Observation and Prediction.

A fundamental presumption behind model-based diagnosis is the notion that if the model is correct, all the discrepancies between observation and prediction arise from (and can be traced back to) defects in the device. Simply put, if the model is right, the device must be broken, and the discrepancies are clues to the character and location of the faults. This is a useful view of the process that will carry us through the first two thirds of the paper.

We will eventually see, however, that it is also a simplified view: The assumption that the model is correct is in fact *necessarily wrong in all cases.* It is wrong in ways that are sometimes quite obvious and sometimes quite subtle. Simply put, a model is a model precisely because it is not the device itself and hence must in many ways be only an approximation. There will always be things about the device that the model does not capture.

The good news is that the things the model fails to capture may have no pragmatic consequence. A schematic for a digital circuit will not indicate the color, smell, or coefficient of friction of the plastic used to package the chips, but this typically doesn't matter. In theory the model is always incomplete, and hence incorrect, in some respects, but it is a demonstration of the power and utility of engineering approximations that models are often pragmatically good enough.

The less good news comes in situations where the approximation is not good enough. In that case we need to ask the more difficult question of how to do model-based reasoning in the face of an incorrect model. What can be done when both the model and the artifact may have defects? We turn to this later in the paper.

Turning back to the basic problem, the task can be specified slightly more precisely by saying that we are given:

- observations of the device, typically measurements at its inputs and outputs (because these are often easiest to obtain; in fact measurements at any point will do and are handled identically),
- a description of the device's internal structure, typically a listing of its components and their interconnection, and
- a description of the behavior of each component.

The task is then to determine which of the components could have failed in a way that accounts for all of the discrepancies observed. Figure 2, for example, shows a device made from three multipliers and two adders. We know the values at the five inputs; the value at output F was predicted to be 12 and observed to be 10 (observations are noted in square brackets). The value at G is predicted to be 12 and has not yet been measured. The overall task is to use knowledge about the structure and behavior of the components to determine which ones could have produced the discrepancy at F, a process explored in detail in Section 6.

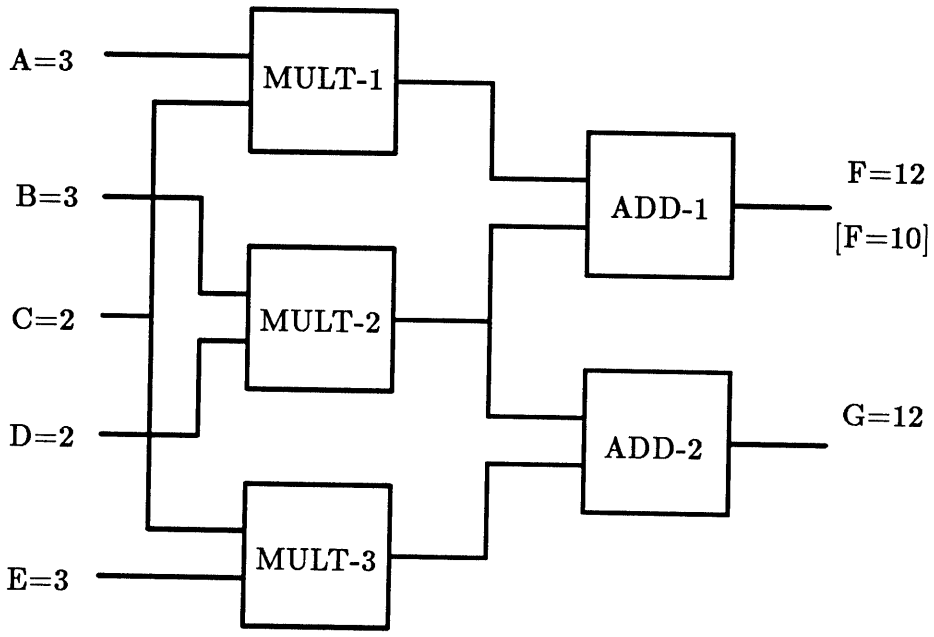


Figure 2: A Common Example.

This approach to troubleshooting has been called by a variety of names in addition to model-based, including "reasoning from first principles" because it is based on a few basic principles about causality, and "deep reasoning," an unfortunate term intended to distinguish it from the associational rules typically used in rule-based expert systems.

Numerous model-based reasoners have been built, exploring a variety of problem domains. The illustrative sample given in Table 1 indicates the growth of interest in the area. Some of the earliest work dates from the mid-1970s, with a considerable growth of interest in the mid-1980s. Much of it has been directed to electronic circuits, both analog and digital, but there have also been applications to problems in neurophysiology, hydraulic systems, and other domains. In the remainder of this paper we use digital circuits as a motivating example, largely because they are a familiar and important application that offers a range of examples from simple to quite complex.

Table 1: Sample Model-Based Troubleshooting Systems.

INTER	[deKleer76]
WATSON	[Brown76]
ABEL	[Patil81]
SOPHIE	[Brown82]
HT	[Davis82]
LOCALIZE	[First82]
IDS	[Pan84]
DART	[Genesereth84]
LES/LOX	[Scarl85]
GDE	[deKleer87]
DEDALE	[Dague87]

The term "model" has been used widely to refer to a range of different things and is somewhat underdetermined. It is thus useful to review briefly some of the different kinds of models that have been used, to get a sense of the character of the information that models have supplied. As noted, the models used in this paper contain information about the structure and correct behavior of the components in the device. Work in [Patil81] describes a medical diagnosis system that used models of behavior without structure, models that indicated how one physiological event in the body could lead to another (e.g., low blood serum pH causes increased respiration, which causes decreased CO_2 concentration). Traditional circuit diagnosis has often relied on fault models, descriptions of the varieties of component misbehaviors

typically encountered. Finally, work in [Pan84] has attacked the problem of dependent failures by building models that capture the behavior of a component when it receives out-of-range inputs and itself begins to malfunction as a result. All of these are varieties of models, so a system built around any one of them could be termed model-based. Within the scope of this paper we are concerned primarily with models of structure and correct behavior.

3 ALTERNATIVE APPROACHES

Since a number of different approaches to diagnosis have been explored over the years, it is useful to consider alternatives to the model-based approach both as a way of setting it in context and as a way of establishing the appropriate circumstances for its use.

One traditional approach has been to use diagnostics, the test programs traditionally used on electronic devices at the end of the manufacturing line, to ensure that the device is capable of doing everything it's supposed to do. A second technique is to build a "fault dictionary" by using simulation and a list of the kind of faults anticipated. The idea here is to simulate the device behavior for every one of the ways in which each individual component can misbehave. Each simulation generates a description of how the entire device would behave if a specific component were broken in a specific way. The overall result is a list of fault/symptom pairs. The list is then inverted so that it is organized by symptom, providing a dictionary that indexes from observed symptom — the surface misbehavior — to one or more underlying faults capable of causing that misbehavior.

Third, we can build programs to do diagnosis by capturing the experience of experts, in the fashion widely used to build rule-based systems that employ empirical associations. Finally, decision trees are a longstanding approach to capturing diagnostic knowledge and offer a way of organizing a set of questions that leads methodically through the process of zeroing in on the faulty component.

Given the diversity of approaches to the problem, why and when does it make sense to use the model-based approach? One way to answer the question is to compare it against the alternatives.

3.1 COMPARED TO DIAGNOSTICS

One problem with traditional diagnostics is that they are misnamed: Diagnostics don't do diagnosis, they do verification. As noted, their job is to ensure that a newly manufactured device will in fact do everything it's supposed to do. There is no misbehavior to diagnose, because there hasn't been any behavior yet. The fundamental task of verification is to exercise all the intended behaviors and make sure that they are all there. That's a different problem.

Model-based diagnosis, on the other hand, is both diagnostic and *symptom--directed*: It starts with the observed misbehavior and works back toward the underlying components that might be broken. As will become clear, whenever the behavior of a device is reasonably complicated, it's much easier to work from a specific symptom back to an underlying fault than to go exhaustively through all the expected behaviors until we find one that's aberrant.

3.2 FAULT DICTIONARIES AND DIAGNOSTICS: PRESPECIFIED FAULT MODELS

As we explore in more detail later, the model-based approach also covers a wider class of faults than both fault dictionaries and traditional diagnostics, because both of those require a fixed, preselected class of relatively simple fault models. For fault dictionaries the task is to select a set broad enough to be useful in practice, yet simple enough that the simulation task is tractable. Writers of diagnostics typically have to settle on a small, fixed class of faults in order to create diagnostics that have acceptable coverage (the percent of possible faults actually detected), resolution (how precisely a detected fault can be localized), and efficiency. In the world of digital electronics the most common choices are the faults known as stuck-at-1 (a node in the circuit always exhibits the value 1) and stuck-at-0, largely because these are easily modeled, simulated, analyzed, and turn out to provide good coverage of other types of faults.

Whatever the faults chosen, the important point is that the fault dictionary creator or diagnostic writer must pre-select a set of things that can go wrong and work from just those possibilities. As will become clear, the model-based approach takes a different view, defining a fault as "anything other than the intended behavior;" one consequence of this view is the ability to cover a wider class of possible misbehaviors.

Fault models do offer two useful abilities. First, as we explore in Section 6.3.1, they can provide an extra degree of specificity to the diagnosis. Where the model-based approach defines a fault by exclusion (anything other than expected behavior), fault models suggest specific misbehaviors that can aid in making the predictions necessary to design further tests.

Second, even though the set of pre-enumerated faults used may be small, it may be adequate for the task at hand. In digital circuits, for example, a large fraction of all faults can be detected (but not diagnosed) by checking just for stuck-ats. Hence two simple fault models turn out to be sufficient for determining that something is wrong (satisfying the verification task); determining the identity and location of the error (diagnosis), however, is more difficult.

3.3 COMPARED TO RULE-BASED SYSTEMS

Traditional rule-based systems have been built by accumulating the experience of expert troubleshooters in the form of empirical associations, rules that associate symptoms with underlying faults and that base those associations on experience with the device, rather than knowledge of structure or behavior. The problem here is the strong device dependence — a new rule set is required for every new device — and the time required to accumulate those rules. To the extent that the knowledge is an encapsulation of experience, a sizable body of experience may be necessary before the patterns emerge.

The issue becomes especially important in dealing with electronic devices, where the design cycle is getting short enough to be comparable to the time required to accumulate a new set of rules. This presents the difficult situation in which the device may be on its way to obsolescence by the time enough experience with it has accumulated to deal with the difficult faults.

The model-based approach is, by contrast, strongly device independent, works from an information source (the design) typically available when the device is first manufactured, and is far more likely to provide methodical coverage. Given a design description for a device, work can begin on diagnosing the device right away. Given a new design description for a different device, work can start on that one just as quickly.

The model-based approach can be less costly to use, because the model needed is often supplied by the description used to design and build the device in the first place. The increasing use of computer aided design and manufacturing also means that those models are increasingly available as explicit descriptions in electronic form, rather than implicit in the head of the designer, or sketched informally on a scattered collection of paper.

The model-based approach is more likely to provide methodical coverage because the model building process supplies a way of systematically enumerating the required knowledge. Systems built from empirical associations capture whatever experience has been encountered to date and offer far less guidance about what may be missing. As a result it is also more difficult to determine the coverage of such a system.

Finally, it may be claimed that rules need not be just empirical associations, they can also be written to take advantage of knowledge about device structure and behavior. But that's just the point: The relevant knowledge concerns structure and behavior. Given that, we ought next to ask what representations are well suited to capturing that information, what representations offer us leverage in thinking about that knowledge. Rules, whether as empirical associations or viewed simply as if/then statements, offer us little or no help in thinking about or representing structure and behavior, or in using such descriptions to do diagnosis. Most fundamentally, they do not even lead us to think in such terms.

In slightly more general terms, the primary question is not whether some existing representation can in some fashion be made to do the task. The primary question is, what is the relevant knowledge, and second, what does that content suggest about appropriate form. We consider such representations in Section 5.

3.4 COMPARED TO DECISION TREES

Decision trees provide a simple and efficient way to write down the sequence of tests and conclusions needed to guide a diagnosis. But the same simplicity and efficiency that is their strength is also an important weakness: They are a way of writing down the "answer" (a diagnostic strategy), but offer no indication of the knowledge used to create that answer. One consequence is a lack of transparency (the tree provides no indication why the diagnosis is what it is) and difficulty in updating (a small change to the device may mean a major restructuring of the tree). Like rule-based systems they are also device specific and must be recreated anew for each new device.

3.5 WHEN NOT TO USE THE MODEL-BASED APPROACH

Comparing the model-based approach to its alternatives provides some indication of its strengths and indications for its use. When does it make sense not to use this approach? The answer can be bracketed by examining problems that are too hard and problems that are too easy to be worth trying this way.

Problems that are too difficult are those involving subtle and complicated interactions in the device, interactions whose outcome is too hard to predict with current modeling technology. Consider, for example, a model of a computer that has been found through experience to have unreliable power supplies. The lack of reliability may arise from a sizable collection of interacting factors, like the heating and insulation patterns, air flow, electric and thermal properties of the materials used to build the power supplies, etc. Predicting such behavior from the design description would very likely be pragmatically impossible, yet summarizing and using it once it has been noticed is quite easy ("if one of these machines is behaving erratically, it's likely to be the power supply"). We are in effect recognizing here that in some cases it's far easier to "let nature do the experiment," watch the outcome, and capture the experience in the form of rules, than it would be to predict the result from first principles.

Future advances in modeling and prediction will extend these limits, but the point remains that, given sufficient complexity, it is easier to let nature do the experiment. Reality is sometimes the cheapest simulator.

Problems that are too easy are those in which the device is so simple that we can model its behavior exhaustively and where the set of faults to be considered is well enough known and well enough understood to be reliably pre-enumerated. In that case it may make sense simply to do exhaustive enumeration and create a fault

dictionary.

We can thus approach the issue of when to use the model-based approach from two dimensions. First, the structure and behavior of the device should be reasonably well known and simple enough to model, but complex enough that exhaustive simulation is infeasible. Second, the set of possible faults should be difficult to reliably enumerate in advance.

4 ORGANIZATION AND VOCABULARY

The discussion in this paper uses several basic ideas as organizing principles. First, we view diagnosis in terms of the three stages of hypothesis *generation*, *test* and *discrimination*. Second, we note that different amounts of knowledge can be brought to bear at each of these stages, producing more or less powerful approaches. Third, the range of programs that can be created by considering different amounts of knowledge at each stage maps out a space of possible program architectures. Finally, and perhaps most interestingly, we claim that this space of architectures captures the current set of programs that have been explored. That is, we can describe all the current model-based systems by characterizing them according to the amount and kind of knowledge they use at each of these three stages.

A number of basic vocabulary terms will facilitate later discussion. By "device" or "system," we mean the entire artifact, e.g., the entire device in Figure 2. By "component" we mean any one piece of it, in this case any of the adders or multipliers. (We may choose to represent wires as components as well; this is an issue of modeling choice discussed later.) By "structure" we mean the way things are interconnected, while "behavior" refers to what any one of these components is supposed to do. We use "discrepancy" to mean any of the differences between the behavior the device is supposed to exhibit (e.g., *F=12*, predicted by the model) and what it is actually doing ([*F=10*], determined by observation). By "suspect" we mean any component identified in hypothesis generation as able to account for a discrepancy (e.g., MULT-1 can account for the discrepancy at *F*). Finally, by "candidate" we mean a component whose malfunction is consistent with all observations (i.e., a suspect that has survived hypothesis testing). When dealing with multiple faults, a candidate may consist of more than one component.

5 DESCRIBING STRUCTURE AND BEHAVIOR

While a number of apparently different approaches to representing structure have been explored, there are several common themes that appear to be widely viewed as good ideas.

- Structure representation should be hierarchical.

Inside any of the boxes in Figure 2, for instance, there are more boxes and wires; look inside those and there are more of the same, until we arrive finally at primitive components. A hierarchical description permits hierarchical diagnosis: Work at the highest level initially until specific candidates have been isolated, then explore inside only those components, since there is no need to examine the substructure of components that are not candidates.

- Structure representation should be object centered and isomorphic to the organization of the device.

By "object centered" we mean that there are data objects corresponding to each of the components in the device; attached to each object is a description of its behavior. The representation should be isomorphic in the sense that the topology of interconnections between the objects should match the interconnections in the device. Hence the object associated with MULT-2, for instance, is connected in the LISP sense to the objects for ADD-1 and ADD-2.

One useful consequence of doing this is that it provides a single, unified representation that is both runnable and examinable. It is runnable in the sense that it can be used directly for simulation: If we supply values for the inputs to MULT-1, for instance, the object corresponding to it will discover that it has enough information to predict its output. It will do so, placing the result at its output, where the information will travel via the connections to the next component in line, which may now continue the process.

The same representation is examinable in the sense that it can be inspected to answer questions about device structure. Because it is in part a graph, questions about connectivity can be answered simply by traversing the representation.

- Behavior can be represented by a set of expressions that capture the interrelationships among the values on the terminals of the device.

The behavior of an adder, for example, can be captured with three expressions (Figure 3), indicating that:

- If we know the values at A and B, the value at C is $A+B$ (the solid arrow in Figure 3).
- If we know the values at C and A, the value at B is $C-A$ (the dashed arrow).
- If we know the values at C and B, the value at A is $C-B$ (the dotted arrow).

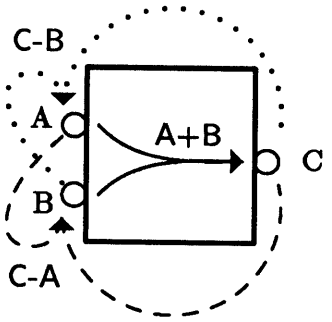


Figure 3: The Behavior Description of an Adder.

While the expressions here are written in algebraic form, the important thing is the knowledge content, not form. Work in [Genesereth84], for example, has explored the use of predicate calculus as a representation for both behavior and structure.

Interestingly these expressions capture both the causal behavior of the device (the bold arrow), as well as other things we can infer about the device (the other two arrows). The first of these indicates how it works, the other two are useful inferences we can make about what must have been at an input, given observations at other terminals. As we'll see, both kinds of information play an important role in supporting diagnosis.

6 THREE FUNDAMENTAL TASKS

We consider next the three fundamental tasks of diagnosis and examine how each has been attacked in the model-based approach, using a variety of different kinds and amounts of knowledge. We consider each in turn, starting with the common simplifying assumption that there is only a single point of failure; as the discussion proceeds we show how some of the techniques can be extended to cover multiple points of failure.

- Hypothesis generation: Given one discrepancy, which of the components in the device might have produced it?
- Hypothesis testing: Given a collection of components implicated during hypothesis generation, which of them could have failed so as to account for all available observations of behavior?
- Hypothesis discrimination: When, as is almost inevitable, more than one hypothesis survives the testing phase, what additional information should be gathered to discriminate among them?

As noted, for the sake of presentation each of these is discussed independently, even though in most implementations they are interleaved for the sake of efficiency. While interleaving offers useful improvements in speed, it produces no fundamental changes to the paradigm.

6.1 HYPOTHESIS GENERATION

The fundamental task here is, given a discrepancy, determine which components might have misbehaved in a way that can produce that discrepancy. Classical AI wisdom tells us that a good generator should be *complete* (i.e., capable of producing all the plausible hypotheses); *non-redundant* (i.e., capable of generating each hypothesis only once); and *informed* (i.e., able to produce few hypotheses that ultimately prove to be incorrect).

In the spirit of proceeding incrementally we consider a sequence of generators from the simplest and least informed, through successively smarter versions that bring additional kinds of knowledge to bear.

The simplest generator, guaranteed to be complete, is one that simply exhaustively enumerates the components in the device. For the device in Figure 2, for instance, the generator simply produces each of the five components one by one. This is trivially complete and not particularly intelligent.

We can improve on this with a succession of observations. For example:

- To be a suspect, a component must have been connected to a discrepancy.

That is, to plausibly explain a discrepancy, the suspect must have in some fashion been involved in it, have contributed to it. Our second generator takes advantage of the insight by traversing the structure description, working from a discrepancy (e.g., at F in Figure 2) to find all components connected to it. In the current case this provides no improvement, since the connections (wires) leading from F reach every component.

We next observe that:

- Devices often have distinguishable inputs and outputs.

This is clearly true for our adders and multipliers (Figure 4) and can be used to constrain the components considered: We need only consider components that are upstream of the discrepancy. In the current example this reduces the set of suspects to ADD-1, MULT-1, and MULT-2.

We can be a bit smarter yet, by observing that:

- Not every input to a device influences the output; there is no need to follow irrelevant inputs upstream.

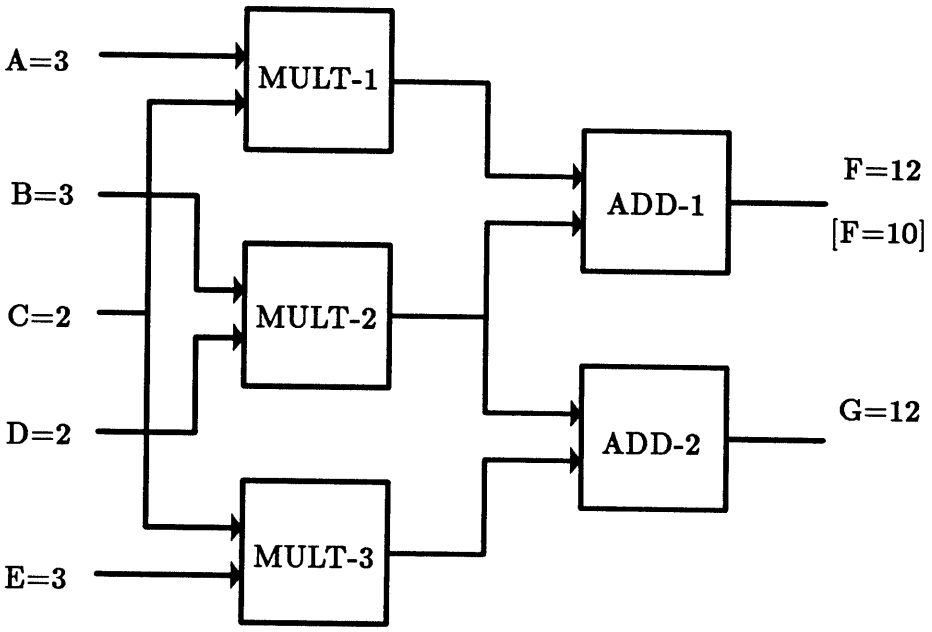


Figure 4: Taking Advantage of Direction of Information Flow.

The easiest example of this is an OR gate whose inputs are produced by two independent collections of components further upstream (Figure 5). Given inputs of 1 and 0, the model for the gate makes the obvious prediction at C. If the actual device is observed to be producing 0 there, three possibilities arise. First, the OR gate itself may be broken. Second, the gate may be working but input A is 0 rather than 1 and the problem lies further upstream in that direction, so we should continue tracing that way.

The third possibility, however is problematic: It is contradictory to believe that the OR gate is working but that the problem lies further upstream of B. No matter what's going on upstream of B, if the OR gate is working, that is not going to account for the observed behavior. As a result we need not consider any components upstream of this point. More generally, the hypothesis generator can use knowledge about component behavior to determine which inputs are irrelevant and avoid tracing back through those.

Finally, we can observe that

- Information from more than one discrepancy can be used to further constrain suspect generation.

When there is more than one discrepancy, we can generate a set of suspects for

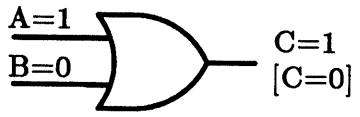


Figure 5: Not Every Input Influences the Output.

each, then (assuming a single point of failure) intersect them, possibly reducing the number of suspects generated. Consider Figure 6, as an example. Tracing back from the discrepancy at F yields ADD-1, MULT-1, and MULT-2 as candidates; tracing from G yields ADD-2, MULT-1 and MULT-2. Assuming a single point of failure, the suspects lie in the intersection of these two sets.

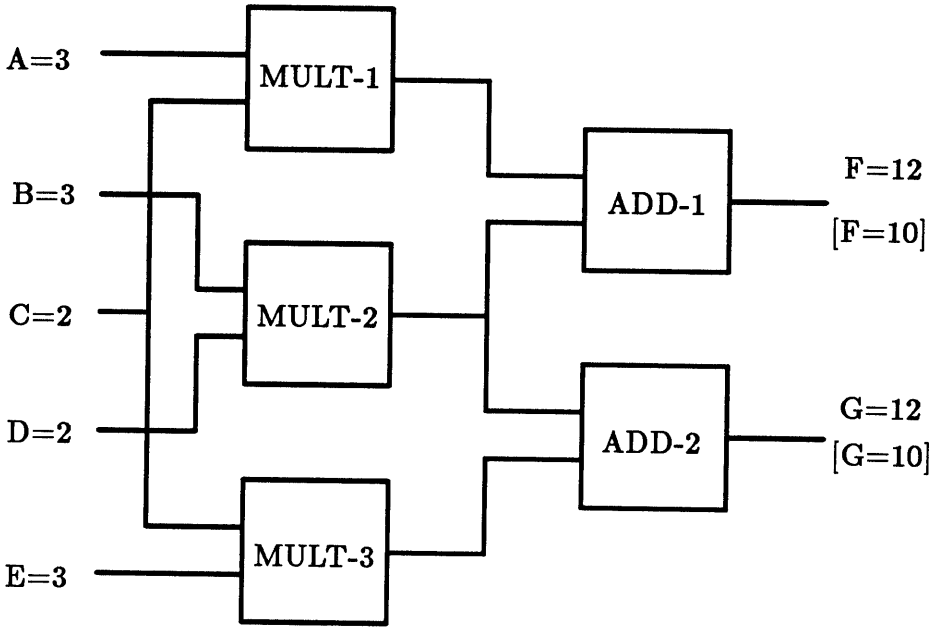
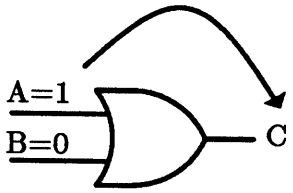


Figure 6: Polybox with Discrepancy at F and G.

This scheme is easily elaborated to deal with multiple points of failure by recognizing that the generalization of intersection in this case is set covering: We are trying to find a subset of components that accounts for (covers) all the discrepancies. To deal with the situation in Figure 6, for instance, we might select MULT-1 from the first discrepancy and ADD-2 from the second, yielding a hypothesized pair of faults that covers all the discrepancies.

6.1.1 Machinery

One brief diversion into mechanism will make clear how to do this kind of reasoning easily and efficiently. The basic insight is to have the simulator record "reasons" as well as values. When the simulator predicts 1 at C, for instance, it records both that value and the expression from the behavior model for the component that produced the value (Figure 7). In this case the simulator would indicate that the value at C is 1 and the reason is *E1*.



E1: If A=1 then C=1
E2: If B=1 then C=1
E3: If A=0 and B=0 then C=0

Figure 7: Recording Reasons as Well as Values.

This simple mechanism offers an easy way to determine which inputs to a component were relevant to its output, further constraining the search for hypotheses. All we need do is inspect the simulation record to determine what expression was used to predict a value, then inspect that expression to determine which inputs it used.[1] In Figure 7 for example, expression *E1* uses only A, hence we need never consider hypotheses upstream of B.

This is a somewhat simplified but essentially correct view of the machinery in most model-based simulators in use today. The general notion is to have the simulator keep track of dependency records that indicate what information was used to determine each new value; generating candidates can then be done simply by tracing back through the dependencies.

A slightly more elaborate example will demonstrate why this technique can be very useful. Figure 8 shows a collection of gates with arrows indicating the records the simulator has kept as it made its predictions. Given a discrepancy at the output, the task of generating a complete, non-redundant and constrained set of hypotheses becomes simply a process of following the trail of electronic bread crumbs back along

[1] Alternatively we could simply record which inputs were used. The scheme given is slightly more general, since the reasons can be useful in other ways, e.g., as a basis for explanation, and the inputs can be determined from them.

the reasons. Part of the overall insight here is that by using a reasonably sophisticated simulator — one that propagates reasons as well as values — the hypothesis generation task becomes relatively simple and straightforward (SOPHIE [Brown82] provided one of the earliest examples of this approach).

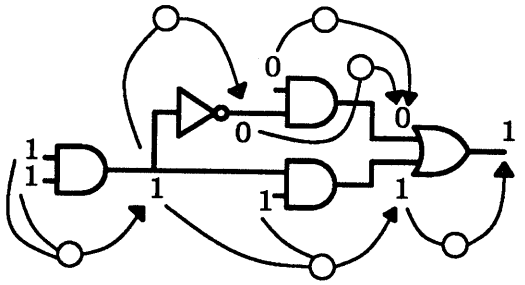


Figure 8: Dependency Traces Left by the Simulator.

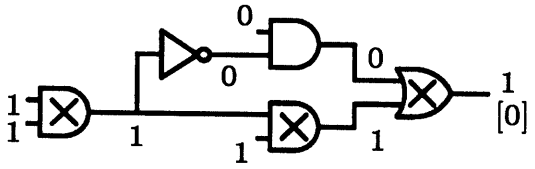


Figure 9: Candidates Selected by Tracing Back through the Dependency Traces.

6.2 HYPOTHESIS TESTING: A SIMPLE TECHNIQUE

In the second basic task of diagnosis — hypothesis testing — the goal is to test each suspect to see if it can account for *all* the observations made about the device. One simple approach is to use fault-model simulation on the suspects produced by the generator (as for example in [Brown82] and [Pan84]). We enumerate all the ways each specified component can malfunction, then simulate the behavior of the entire device on the original set of inputs under the assumption that the candidate is malfunctioning in the way specified. If the overall predicted behavior is inconsistent with the observations, the hypothesis can be discarded; hypotheses accounting correctly for the observations pass the test and are retained. The result is a set of hypotheses specifying how each suspect may be malfunctioning.

One interesting additional inference can be made if we believe that the pre-enumerated list of misbehaviors is *complete*: If none of the misbehaviors hypothesized

for a component matches the observations, that component must be working correctly in the current situation and can be exonerated. It may or may not be working perfectly in all circumstances, but it is not causing the current set of discrepancies and we will have to look for the fault elsewhere.

If the misbehavior list is not believed complete, the component cannot be exonerated, since it may be misbehaving in some as yet unknown fashion. In this situation we may end up with two categories of suspects: those for which a hypothesized misbehavior matches the observations and those that may be failing in an unknown way. In that case it may make sense to treat the first category as more likely, falling back on the second only as necessary.

6.3 HYPOTHESIS TESTING: MORE ADVANCED TECHNIQUES

Three other slightly more advanced techniques use knowledge about device behavior to generate and test hypothesized candidates, yet do not require a pre-enumerated set of misbehaviors.

6.3.1 Constraint Suspension

Constraint suspension [Davis84] tests whether a suspect is consistent with all the observed behaviors of the device. The basic idea is to model the behavior of each component as a set of constraints, and test suspects by determining whether it's consistent to believe that only the suspect is malfunctioning. That is, given the known inputs and observed outputs, is it consistent to believe that all components other than the suspect are working correctly?

Consider the standard circuit as an example, in a situation in which the inputs are as shown in Figure 10 and where values at both outputs have been measured, yielding a discrepancy at F and the predicted value at G. The behavior of each component is modeled as a set of constraints of the sort shown previously in Figure 3; Figure 10 shows the entire device with the constraint network sketched in.

This network and set of values is clearly inconsistent. That is, given this set of constraints, if the values shown were inserted at the inputs and outputs, some constraint would soon encounter an inconsistency, i.e., attempt to fire and record a value at a node where there was already a different value recorded. Since constraints can propagate either from inputs to outputs or from outputs to inputs, the inconsistency might occur anywhere in the network (at the outputs, the inputs, or an interior node). The important point is that the network would report an inconsistency somewhere.

The traditional approach to handling inconsistencies in constraint networks is to find a value to retract. Here, however, we are sure of the values (the inputs sent in and the outputs measured); we are, however, unsure of the component behaviors. Constraint suspension thus takes the dual view: Rather than looking for a value to

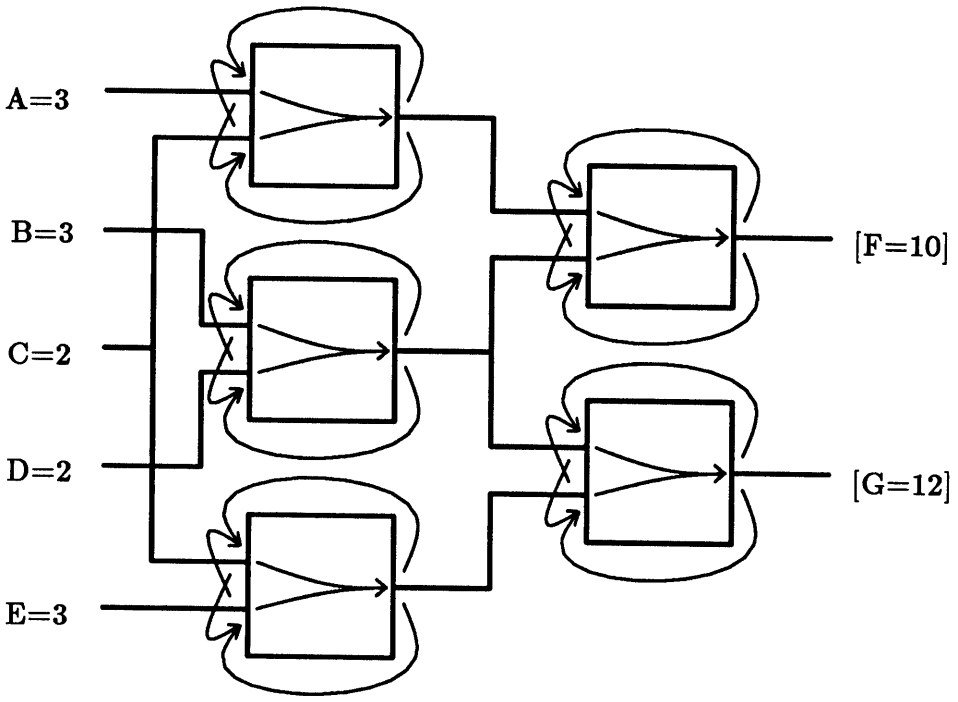


Figure 10: The Constraint Network View of the Device.

retract, it considers which constraint to retract to remove the inconsistency.

To put this back in hypothesis testing terms, recall the basic question stated above: Given the available observations, is it consistent to believe that all components other than the suspect are working correctly? "Working correctly" means the component is behaving as the model predicts; this is simulated by having the corresponding constraint "turned on." To say something is a suspect, by contrast, is to indicate that we don't know what it's doing, what its behavior is. In that case the most conservative stance is to retract all assumptions about its behavior. This is simulated by suspending its constraint, i.e., removing it from the network temporarily. Figure 11 shows the situation when testing the hypothesis that MULT-1 may be at fault.

Hypothesis testing is thus accomplished by suspending the constraint for the suspect, leaving in place the constraints for everything else, then putting in the observed values and allowing the (reduced) constraint network to run to quiescence. If it does so without encountering an inconsistency, we get two interesting pieces of information. First, we know that the current suspect is in fact consistent with all the observations, i.e., there is some behavior for it that can account for all the observations. Second, the constraints often propagate values to the terminals of the suspect, supplying in-

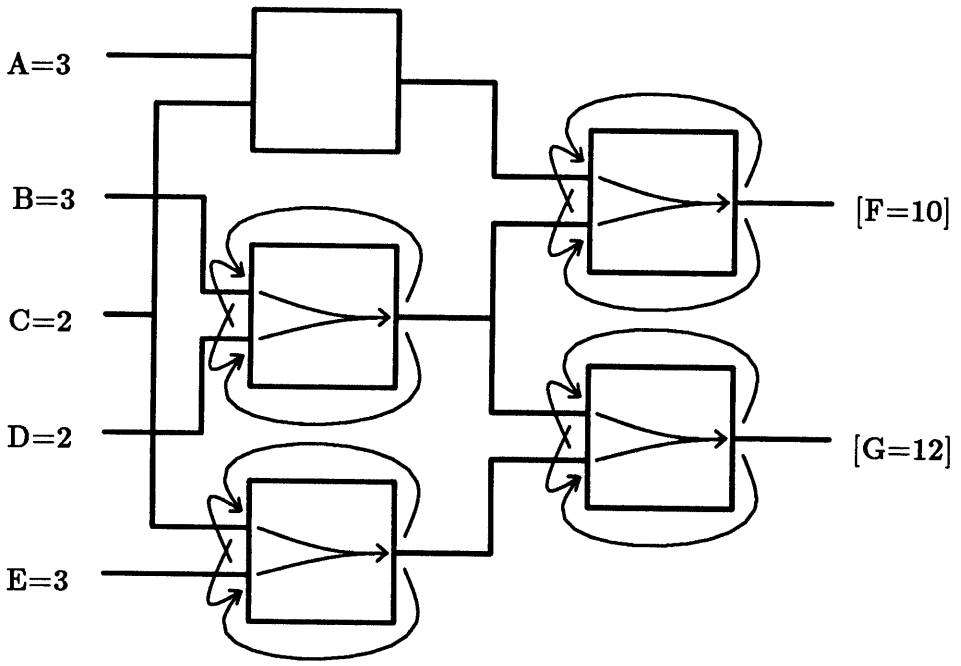


Figure 11: The Network with the Constraint for MULT-1 Suspended.

formation about how it must be misbehaving. For example, the constraint network in Figure 11 will eventually determine that MULT-1 is a consistent candidate that could be multiplying 2 and 3 to produce 4. This ability to infer component symptoms is clearly dependent on the ability to propagate "backwards," in this case inferring the upper input of ADD-1 from its output and lower input.

If the network is still inconsistent even with the suspect's constraint suspended, the current hypothesis can be rejected, exonerating the suspect: There is no set of assignments to the terminals of the suspect consistent with the observed values and the constraints currently in effect. This occurs when testing MULT-2, one of the three suspects produced by hypothesis generation for the situation in Figure 10. With only the constraint for MULT-2 suspended, there is no set of assignments to its terminals that is consistent with the inputs and outputs observed. It can thus be rejected.

There are several interesting properties of this technique. First, as noted, it not only indicates whether or not something is a consistent candidate, it can often specify the symptoms at the terminals of that component.

Second, the power of hypothesis testing and its ability to infer symptoms are dependent on the power of the propagation machinery. Current constraint systems are "local" in the sense that they propagate values through one component at a time,

at each step solving one equation in one unknown. This style of propagation can stall when it encounters situations requiring more sophisticated algebra (e.g., solving two equations in two unknowns). Such situations are relatively common in domains with non-directional components and can arise in domains with directional components in structures that have reconvergent fanout (i.e., a signal that branches and then rejoins at the inputs to a component).

The complexity of the algebra required depends on both the vocabulary used in the behavior language and the interconnection topology of the devices; it can quickly grow quite difficult. Some research has attacked the problem by propagating symbolic expressions rather than numbers (e.g., [Sussman80]), exhaustive enumeration has also been explored where ranges are finite. If propagation does stall, the system will judge the candidate consistent because no contradiction was derived, even though there may in fact be one. Other work, relying on direct symbolic manipulation of expressions (e.g., [Genesereth84], [Scarl85]) encounters similar problems where their symbolic solution methods are not complete.

Some candidates accepted as a result of stalled propagation are valid; in those cases there is no adverse consequence. Even when an invalid candidate is accepted, however, the only consequence is that the candidate set is larger than it should have been. The diagnosis is thus somewhat less precise, but at least no valid candidate is overlooked.

Third, where many traditional techniques require specifying how a component can fail, the reasoning above simply withdraws any commitment to how it might be behaving. That is an interesting property of model-based reasoning in general, not just the constraint suspension approach: Something is malfunctioning if it's not doing what it's supposed to, *no matter what else it may be doing.* As a result there is no need to pre-specify how the component might fail, a fault is any behavior that doesn't match expectations.

It is in that sense that the model-based approach, using a model of correct behavior, covers a broader class of faults than traditional techniques that require pre-specified fault models. Note for instance, that the device in Figure 11 may be misbehaving because the wrong kind of chip was inserted into the socket where the multiplier was supposed to go. In that case there is no simple model for the misbehavior and no plausible way to diagnose it in the traditional fashion. Yet the model-based approach handles this case because it need only observe that the component isn't doing what it's supposed to do.

The fault model approach falls short in this case because its models combine both physical and logical plausibility. The model-based approach by contrast deals only with logical plausibility, asking simply whether there is *any* set of values the component might display that can account for all the observations. The technique, by design, does not ask whether that set of values is in fact physically plausible.

As a result it can suggest candidates that, while logically plausible, are in fact physically unrealizable, requiring a second pass to filter them out. This can, however, be an advantage because physical plausibility is technology-specific. A broken wire, for instance, can manifest differently depending on the technology; in TTL logic, for instance, it will appear as a high. Embodying this knowledge separately can both ease the initial construction task and reduce the difficulty of applying model-based reasoning to a new domain.

The traditional use of fault models can also be seen as trading off breadth for specificity: By committing to a pre-specified set of set of possible failures, we can gain in return greater specificity in the diagnosis. In the case of MULT-1, for instance, the model (of correct behavior) approach can say only that the component is multiplying 2 and 3 to get 4, while the fault model approach might indicate as one possibility that the 2-bit of the output is stuck at 0 (turning 6 into 4).

The model-based approach thus supplies a behavioral description of the misbehavior for this specific case, and, by design, says nothing about what the malfunctioning component would do with any other inputs. This permits it to cover a broad variety of possible failures. The fault model approach, on the other hand, pre-commits to a specific set of malfunction mechanisms and as a result can be more specific about what is wrong and can provide the basis for predictions of misbehavior for other inputs (e.g., if the 2-bit is stuck at 0, MULT-1 should produce 0 when given inputs of 2 and 1). The tradeoff available thus asks whether we are willing to pre-specify the faults and believe that the list is complete enough; if so fault models might offer useful power.

Finally, we have so far been dealing with the single point of failure assumption. Multiple points of failure are trivial to check using constraint suspension: To check for a pair of failures, for instance, suspend the two corresponding constraints, then proceed as before. Generating multiple fault hypotheses efficiently, however, is somewhat more difficult; no simple extension of constraint suspension offers much leverage on this inherently exponential problem. This issue will resurface when we explore GDE [deKleer87], below.

6.3.2 Combining Generation and Test

Two systems — Dart [Genesereth84] and GDE [deKleer87] — integrate hypothesis generation and testing sufficiently that when viewed in terms of generate and test they are best considered systems in which all of the testing knowledge has been integrated into the hypothesis generator.

6.3.2.1 DART

The DART system illustrates the use of predicate calculus as a mechanism for model-based reasoning, with structure and behavior represented as axioms. The connection of MULT-1 to ADD-1, for instance, would be represented as

```
CONN(OUT(1, MULT-1), IN(1, ADD-1))
```

indicating that the first (only) output of MULT-1 is connected to the first input of ADD-1. Part of the behavior description of an adder would be

```
IF    ADDER(a) AND VAL(IN(1, a), x) AND VAL(IN(2, a), y)
THEN  VAL(OUT(1, a), x+y)
```

indicating that, if **a** is an adder with inputs **x** and **y**, its output will be **x+y**.

DART views diagnosis as a form of constrained theory formation. Starting with a set of observations of device misbehavior, the goal is to produce a description of its (faulty) structure. Given only the observations, the task would be the same as designing a device that exhibited the observed behavior. The design description is used to constrain the process by forcing the system to consider only propositions from the design description or their negation. A diagnosis in DART is thus a deduced proposition like

```
(OR (NOT (MULTIPLIER MULT-1)) (NOT (ADDER ADD-1)))
```

indicating which component might be misbehaving.

To arrive at these deductions the system uses a technique called resolution residue, a variation on resolution that works as a direct proof procedure (rather than a refutation method), guided by a number of strategies like unit preference for reducing the number of useless deductions. Details of the process can be found in [Genesereth84]; at the knowledge level the deductions work much like the dependency tracing mechanism reviewed earlier, except in this case dependencies are deduced as needed (via the behavior descriptions) rather than automatically recorded when doing simulation. DART also uses the same resolution residue mechanism for test generation, providing a certain economy of machinery.

Among the limitations in this approach are the occasional difficulties in expression logic can present. The single point of failure assumption in [Genesereth84], for example, requires five distinct axioms for a five component device, each stating that if one is broken the other four must be working. Further work in [Ginsberg86] has demonstrated that reasoning from counterfactuals can produce a notion of minimal faults, at some increase in the complexity of the modeling and inference task.

One of the advantages of logic as a representation and reasoning mechanism is the potential for demonstrating the completeness of the inference procedure. While this can be useful, it does not not imply that the resulting diagnostic process is complete. There are at least two sources of difficulty. First, as noted in [deKleer87], completeness of the inference procedure does not imply completeness of the prediction machinery. As one example, behavior descriptions for analog devices can involve higher-order differential equations; producing exact values for predictions in such devices means solving solutions of such equations, yet no general purpose technique exists.

Second, all of the inference, i.e., all of the candidate generation, is done with respect to the device model supplied, and completeness of the inference machinery is quite distinct from the completeness of the model. Simple examples of the problem arise when axioms are accidentally omitted; more subtle instances arise because, as we argue below, the model is necessarily incomplete. Thus while it can be useful to demonstrate completeness of the inference machinery with respect to the model, completeness of the diagnostic process is a distinct issue. Indeed we argue below that the bulk of the work and difficult problems are in the modeling.

6.3.2.2 GDE

The GDE system [deKleer87] provides a single mechanism for generating both single and multiple fault hypotheses, and presents a carefully constructed strategy for measurement selection. At this point we deal with a few of the ideas for hypothesis generation, illustrating the basic notions with a few simple examples; we return to the issue of measurement selection when discussing hypothesis discrimination in Section 6.5.1. A detailed picture and additional examples of GDE can be found in [deKleer87].

One important enabling technology for GDE is the use of an assumption-based truth maintenance system (ATMS), i.e., one that propagates both values and assumptions. The reasoning begins much like that done previously, with some difference in the record keeping. In Figure 12, for example, if we assume that MULT-1 is working, we can use its behavior description to predict the value at X, then record both the value and the set of underlying assumptions (in parentheses). Values that have been measured (in this case inputs and outputs) have no assumptions, indicated by the null set.

A particularly interesting event occurs when there are two contradictory predictions for the same point in the circuit, as in Figure 13, which shows the next step in the reasoning. The value at X is also predicted to be 4, this time using the (measured) value at F, the prediction at Y, and the assumption that ADD-1 is working. Note that assumptions accumulate: The prediction $X{=}4$ carries all the assumptions it relies on.

This is interesting because of the inference that can now be made: If all three assumptions so far were true, (i.e., MULT-1 *and* MULT-2 *and* ADD-1 were all working),

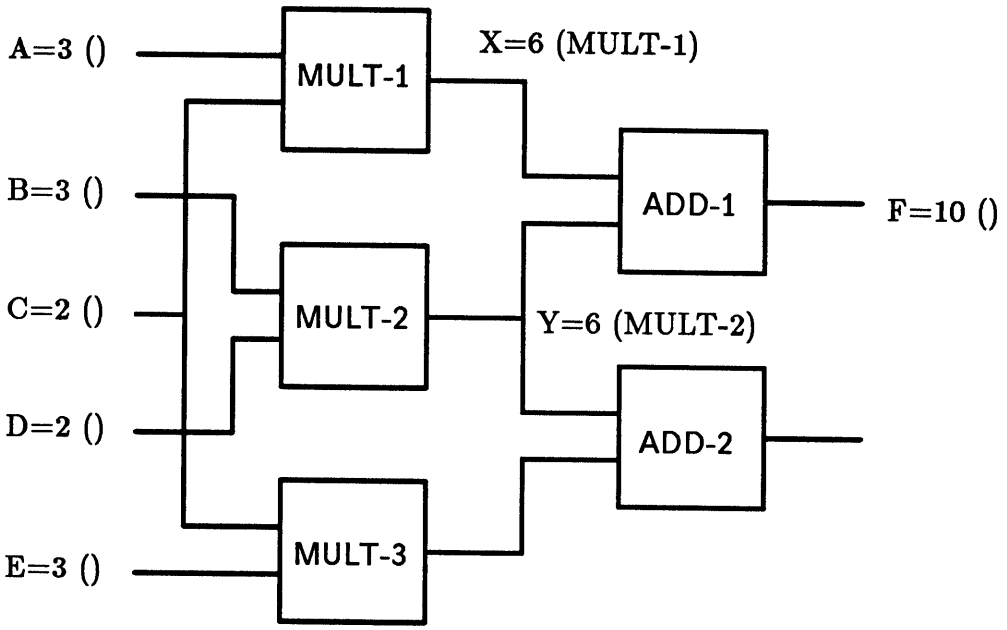


Figure 12: Values and Records Produced by an Assumption-Based TMS.

there is an unavoidable contradiction — two different values at X. Taking the obvious step, we turn that around, inferring that one of the three assumptions must be wrong (i.e., one of the three components is not working correctly).

This is the process of constructing "conflicts": Whenever there are two different predictions for the same place in the circuit, collect all (i.e., take the union) of the assumptions underlying the conflicting predictions. The resulting conflict indicates that at least one of the components in it must be malfunctioning.

Continuing the propagation process in Figure 13 eventually yields a second conflict as well:

C1: (MULT-1 MULT-2 ADD-1)
C2: (MULT-1 MULT-3 ADD-2 ADD-1)

The second step in GDE is to generate a set of candidates that deals with all of the conflicts. MULT-1, for example, is a candidate because it can account for both C1 (one of (MULT-1 MULT-2 ADD-1) must be broken), and C2 (one of (MULT-1 MULT-3 ADD-2 ADD-1) is broken.) Since a single component is capable of accounting for all the conflicts, one of the hypotheses in this case happens to be a single point of failure. ADD-1 is a similar hypothesis; single point of failure hypotheses are produced by intersecting the conflicts.

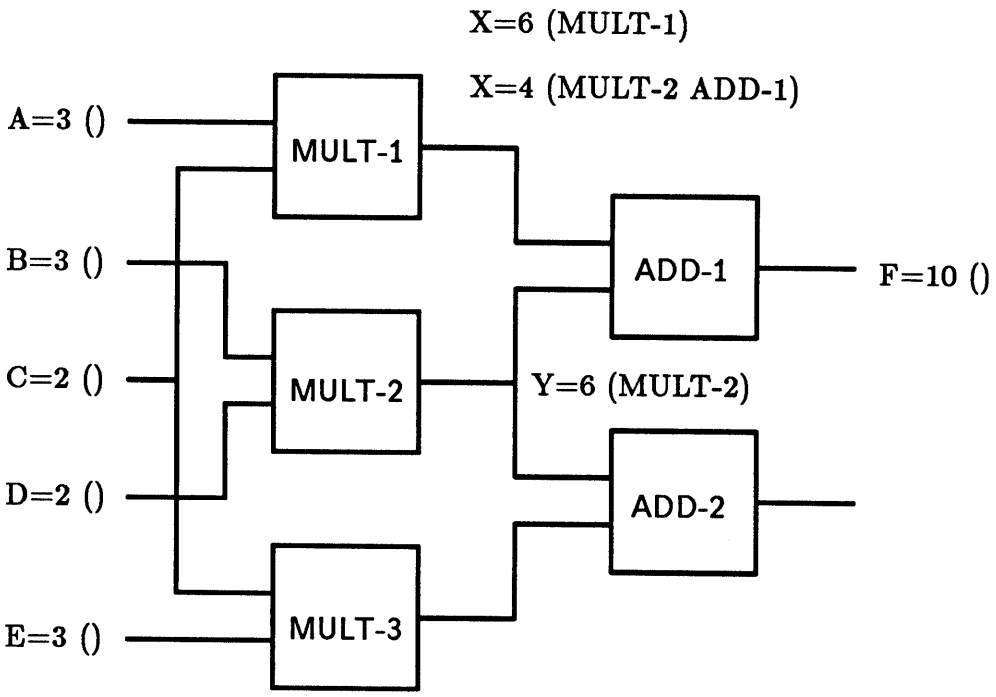


Figure 13: One More Step in the Propagation.

Accounting for conflicts can be viewed more generally in mathematical terms as set covering: We want a collection of components that covers all the conflicts. Singleton covers like (MULT-1) produce single point of failure hypotheses, multiple point of failure hypotheses are generated by larger set covers like (MULT-2 ADD-2), which takes MULT-2 from the first conflict and ADD-2 from the second.

This process is fairly intuitive, but it can be expensive — computing set covers is in the worst case exponential. One way to reduce the potential impact of this complexity is to use the notion of minimality in both conflicts and hypotheses. The basic intuition is the same in both cases: Any superset of a conflict is also a conflict; any superset of a hypothesis is also a hypothesis. GDE uses this to reduce the amount of work it does by generating and maintaining only minimal conflicts (i.e., no subset of one is also a conflict) and minimal hypotheses (i.e., no subset of one also covers all the conflicts). By doing this, the system need never examine any non-minimal conflict or hypothesis, saving a substantial amount of work. While the fundamental exponential character of the problem has not changed, the effect has been reduced, enabling the system to handle problems larger than might otherwise have been possible.

The candidate generation part of GDE thus offers an efficient and intuitive mech-

anism for generating both single and multiple fault hypotheses in a unified approach. The system also offers a degree of mechanism (and hence domain) independence, because the diagnostic process in GDE is separated from the machinery used to predict behavior (the ATMS).

6.4 HYPOTHESIS TESTING VIA CORROBORATIONS

It is useful to discuss briefly the notion of corroborations, the situation in which a measured value matches (corroborates) the prediction at that point. Using corroborations to do hypothesis testing is potentially useful, but must be approached with caution. The basic intuition is seductive: Having seen that any component involved in a discrepancy is a suspect, there is unfortunately a great temptation to construct an overly simplistic dual principle — any component involved in a corroboration must be innocent.

Figure 14 illustrates the difficulty in an example that has a discrepancy at F but a corroboration at G, where the observed value matches the predicted value. Straightforward topological tracing back from F yields the usual candidates (ADD-1, MULT-1, MULT-2). We are now, however, tempted to say that since the measurement at G matches the prediction, all components involved in that corroboration (i.e., MULT-2 MULT-3, and ADD-2) can be exonerated.

The seductive part is that it works in this case and some others, leading at times to unjustified optimism that it is valid in general. The difficulty is illustrated by the simple counterexample in Figure 15, in which ADD-2 has been replaced by a component that computes the maximum of its inputs. Once again there is a conflict at F and a corroboration at G, yet this time the exoneration is incorrect: MULT-2 might in fact be broken, producing 4 as its output.

In general the problem is fault masking, the situation in which a device receives incorrect inputs, yet produces the output that would have been expected with the correct inputs, masking further effects of the fault. Consider MAX-1 for instance: If it receives incorrect inputs of 6 and 4, it still produces the expected output, 6, that would have resulted from the correct inputs (6 and 6).

Fault masking can arise in several ways. Any component that can be insensitive to one of its inputs (e.g., MAX-1) can mask a fault on that input even when working correctly. Multiple points of failure can produce the problem, when one broken component outputs an incorrect value, but a second broken component further downstream masks some of the effects by producing the expected value despite the incorrect input. Finally, even with a single point of failure, the phenomenon of reconvergent fanout can produce fault making.

In Figure 16, for example, component B computes the square of its input, component C computes $16 - 5x$, and ADD-1 is an adder. Component A is supposed to produce 3, which should eventually result in ADD-1 producing 10. If A instead

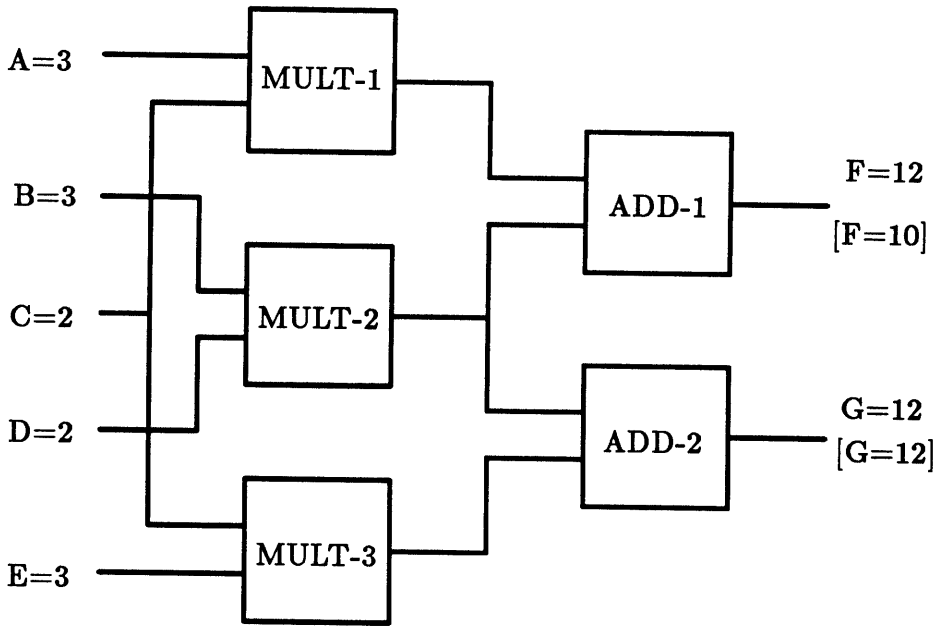


Figure 14: The Standard Example with a Corroboration at G.

incorrectly produces 2, B, working correctly, will produce 4, while C also working correctly produces 6. The final output at the adder is then the expected 10, despite the single fault present in the circuit. If the signal from A fans out to other places, its error would be manifest elsewhere, yet if we naively trace back from the corroboration at ADD-1 we would incorrectly exonerate A.

One important reason to be wary about corroborations is thus the number of and subtlety of the phenomena that can cause fault masking and invalidate corroborations as a heuristic.

A second reason is the asymmetry in the consequences of mistakes in hypothesis generation and in hypothesis testing. If the hypothesis generator is overzealous, we may have more hypotheses to test than are logically necessary, but the system will, at worst, be less efficient than it should have been. Overzealous exoneration, on the other hand, can cause the system to arrive at the wrong answer by ruling out a valid candidate. As a result, it may be plausible if desired to be aggressive with respect to hypothesis generation, but it general it is useful to be more cautious about hypothesis testing.

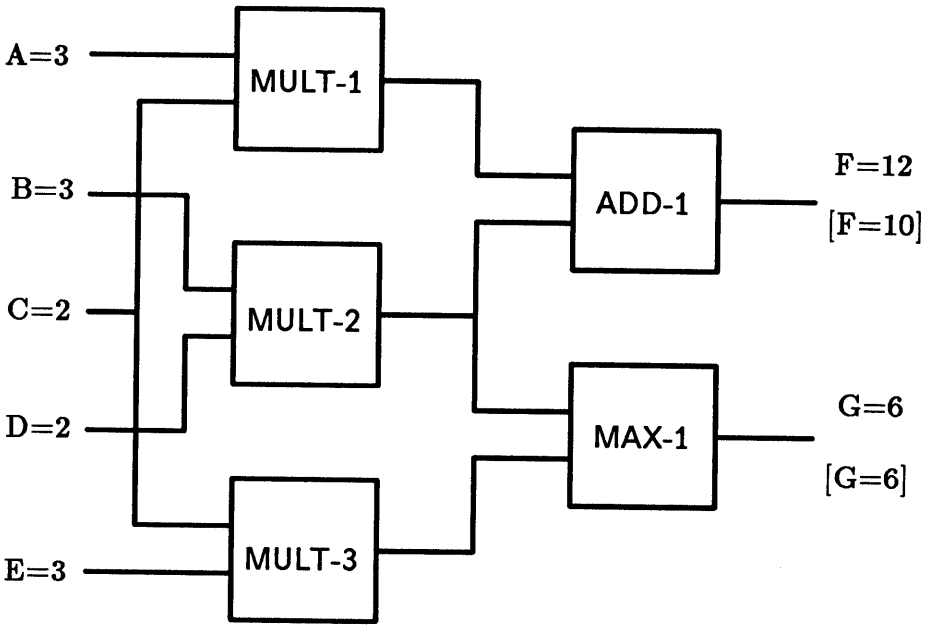


Figure 15: Counterexample Showing that Corroboration is not Valid in General.

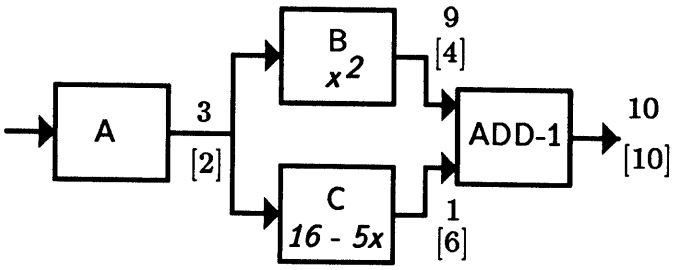


Figure 16: Reconvergent Fanout Can Produce Fault Masking.

6.5 HYPOTHESIS DISCRIMINATION

Having examined generation and testing, we next consider hypothesis discrimination, where the fundamental problem is how to distinguish among the hypotheses, when, as is almost inevitable, more than one survives testing. Distinguishing among competing hypotheses involves gathering new information about the behavior of the device, either by (i) making additional observations (probing), or (ii) changing the inputs and making observations in that new situation (testing). In both cases the goal is to gain the most information at the least cost.

6.5.1 Probing

In considering probing strategies we proceed as before in steps from the most elementary approach to successively more sophisticated techniques. The simplest approach is to use only structural information to generate the set of all possible probe locations and pick any place that has not been measured previously. Refinements to this include (i) using knowledge about component behavior (ii) using knowledge about expected failure rates, and (iii) trying to find the measurement that will lead to the shortest sequence of probes.

6.5.1.1 Using Structure and Behavior

Perhaps the most straightforward and widely used approach is the guided probe. The fundamental idea is to start at the discrepancy and follow it upstream to a component that has an incorrect output but whose inputs are correct. If the component receives valid information but produces a bad result, it must be the culprit. Given the discrepancy in Figure 17 at F, for instance, we probe A and Z next, since if these are observed to have their predicted values, MAX-1 must be broken. If Z has any value other than 5, we probe upstream at both B and Y to see if they are 1 and 4 respectively, and so forth until we find the culprit.

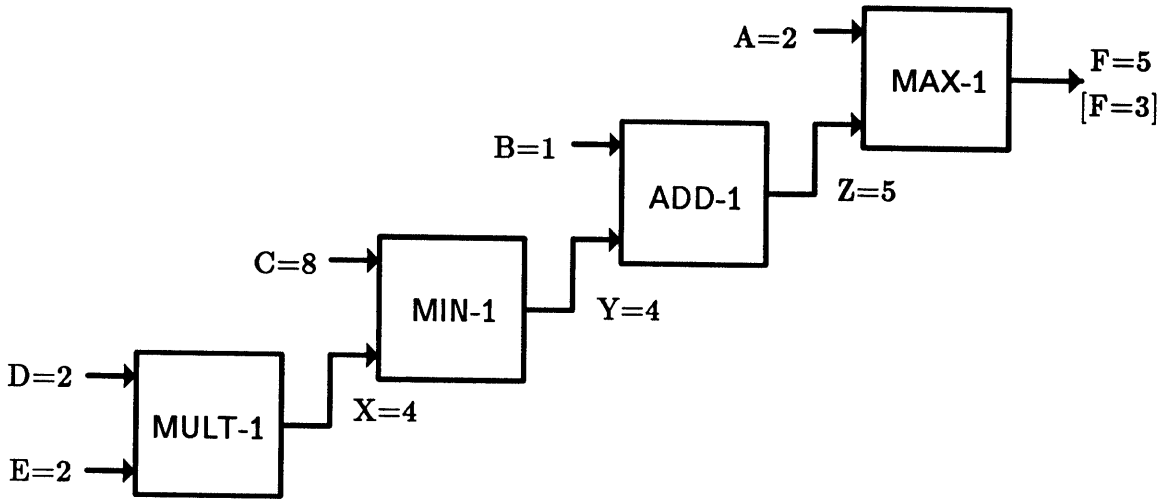


Figure 17: Guided Probe Example.

6.5.1.2 More Sophisticated Use of Behavior

Note that it was not in fact necessary to probe at A, since a discrepancy there alone could not have produced the observed value 3 at F. The guided probe technique can be

extended to use information about component behavior to reduce the probes needed; [Breuer76], for example, shows how it can be applied to Boolean digital circuits. The reasoning involved is similar to that described earlier for using behavioral information to constrain hypothesis generation.

The guided probe approach is appealing in its simplicity and intuitive clarity. It is also, however, a linear time search, which, with even a little cleverness, can be turned into a much more efficient binary search. In the current example, for instance, simply examining the topology of the device makes clear that Y is a more effective probe. If the value there is bad, half the components are exonerated — all those downstream from it. In general the "half split" probe point can be found by considering for each probe point the value that would be predicted there given each suspect; the favored probe is the one that splits the set of current suspects. Figure 18 shows that Y is the best probe: Y will be 2 if MULT-1 or MIN-1 are broken, and 4 if ADD-1 or MAX-1 are broken; either outcome thus rules out half the suspects. Ideally, the process of cutting the search space in half can be continued at each step, producing the traditional binary search, with its potential increase in speed from linear in the number of suspects to be discriminated, to logarithmic. The maximal advantage arises in cases like this with a linear cascade of components, with somewhat less (but still useful) improvement in other cases.

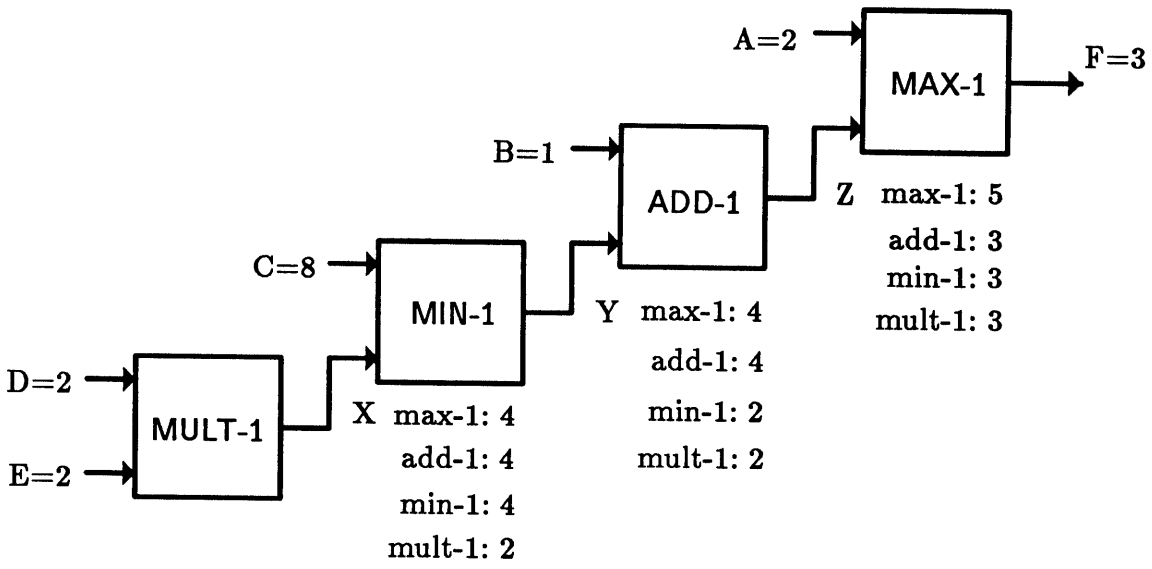


Figure 18: Half Split Strategy Example.

6.5.1.3 Using Failure Probabilities

The example above is particularly easy because one probe is clearly more informative than the others. In more realistic cases several places may be equally informative. If, for instance, we apply our methods so far to the example in Figure 19, X and Y turn out to be equally informative.

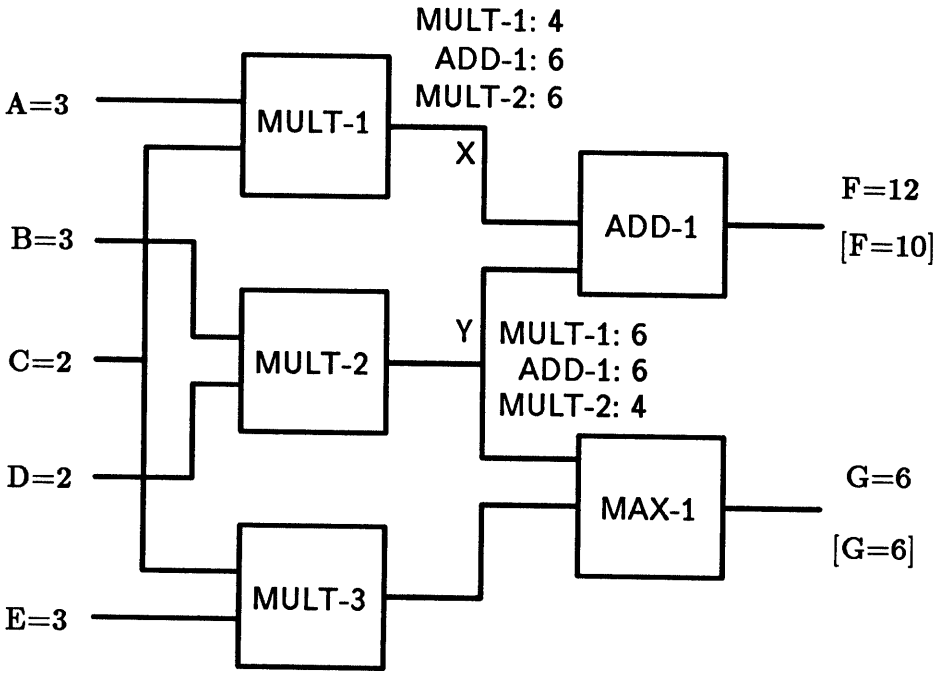


Figure 19: Two Equally Informative Probes.

In the event that MULT-1 and MULT-2 happen to be implemented using different chips that have different reliability histories, it would make sense to "play the odds" by probing at the place that has the greatest chance of having an incorrect value. If MULT-2, for instance, has a much higher *a priori* likelihood of failure than MULT-1, it would be more efficient in the long run to try probing at Y first.

While this example uses failure probabilities to help select among probe points that are indistinguishable using value predictions, the two are independent sources of information. We can in general combine information from predictions (about how discriminating a probe can be) with information from failure probabilities (about how likely that probe will encounter an incorrect value), to yield a measure of how informative a particular probe is likely to be.

6.5.1.4 Selecting Optimal Probes

We have thusfar used information about predictions and failure probabilities to look only one measurement ahead. The analysis in the previous section, for instance, considered what single measurement looked best. A more powerful strategy would determine what *sequence* of measurements was likely to be the most effective, since, as with any search problem, the best path is not always clear from a one-step lookahead.

One obvious approach is exhaustive lookahead: The current predictions indicate the potential places to probe first, we can then make new predictions based on the possible outcome of each of those probes, use that information to determine the set of possible places to probe second, make new predictions based on those, etc., continuing until the sequence of hypothesized measurements would identify a unique fault. This is a classic decision tree analysis and as always the difficulty is the size of the search space.

As with any search problem, the challenge is to find a way of estimating the value of a path without having to explore it to the end. The GDE system takes an information theoretic approach, using the notion of minimum entropy as the basis for its evaluation function (see [deKleer87] for details). Part of the difficulty in applying this idea lies in determining the probability that a particular measurement will have a particular value when not every candidate predicts a value at that point. GDE develops a careful approximation and uses it to select a measurement that is, under a reasonable set of assumptions, optimal in the sense that it minimizes the expected total number of probes.

This approach is well-suited to GDE because the assumption-based TMS that it uses maintains a substantial body of context information that includes the values predicted at each point in the device. Hence little additional machinery is needed to generate and keep track of the required information.

6.5.2 Testing

Testing is the second basic technique for hypothesis discrimination. Here the fundamental idea is to select a new set of inputs to the device that will help reduce the suspect set by providing additional information about the behavior of the device. To remain valid, a suspect has to account for both the original symptoms and the behavior observed in response to the new inputs. As with probing, the difficult task is selecting a set of inputs that is particularly informative.

If the set of tests that can be presented to the device is fixed in advance, the problem of selecting an informative test is essentially equivalent to probe selection. For each test, each suspect (ideally) predicts a certain outcome, hence the best test is the one which splits the set of suspects in half.

If, on the other hand, the set of possible tests is unknown or pragmatically infinite, it is necessary to generate an appropriate test. A simple, sub-optimal technique will

serve to illustrate the basic idea and difficulties: Design a test for each suspect in turn, that is, find a set of inputs that will give two different outputs depending on the condition of that one component. This will serve to determine whether the fault is in the current suspect or among those remaining.

As an example, assume that AND-gate AND-1 in Figure 20 is suspected of malfunctioning, in particular of taking in 1's and sending out 0. We want a set of inputs that will indicate whether that is really how the component is behaving.

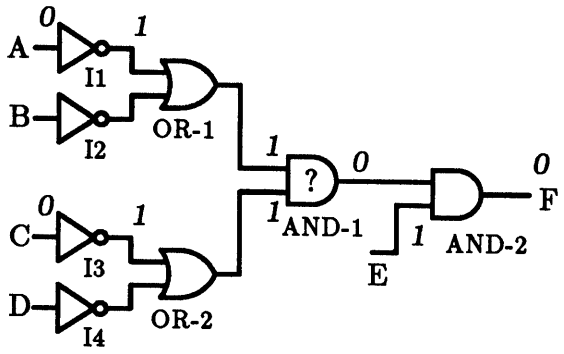


Figure 20: An Example of Test Generation.

To do that we need to get a 1 to both inputs to the gate, then ensure that its output is routed out to where it can be measured. The intuition is straightforward: Work backward from the inputs of AND-1, then forward from the output. We can get a 1 on the upper input by ensuring that OR-1 outputs a 1; this in turn can be ensured if the input to inverter I1 at A is 0. The value at B then does not matter. Similar reasoning from the lower input of AND-1 yields 0 at C. Then in order to ensure that the output of AND-1 can be measured accurately at the device output, we need a 1 at E, the lower input to AND-2. With that input vector it appears that the value at F will determine unambiguously whether AND-1 is malfunctioning in the manner noted.

This style of reasoning is the essence of test generation as traditionally practiced. While the approach is appealing in its intuitive clarity and simplicity, it has important limitations. For our purposes, the most significant limitation is its insensitivity to the presence of other suspects in the device and the resulting insufficient specificity. What if, in the current example, I1 and I3 also happen to be suspects? The test vector selected will generate a value at F that depends on the state of more than one suspect: If the value is incorrect any of the three components may be to blame.

Stated in this fashion the difficulty immediately suggests one plausible remedy: When routing signals through the device, whenever possible route the signal only

through known good components (components that are not suspects). Using this strategy the test generation process would select I2 and I4 to provide the inputs to the OR gates, and end up producing a test that was completely specific, that is, dependent on the condition of only one suspect, AND-1. Work in [Shirley83] describes a system that reasons in this fashion and that produces tests that are as informative and specific as possible.

A second substantial problem in testing arises in circuits with reconvergent fanout. If, for example, the lower input of AND-2 had been attached to input D (rather than having its own input E), the value at D would have entered into two different goals: Establishing the lower input to AND-1 and routing the output through AND-2. It is thus a problem of planning in the face of interacting conjunctive subgoals, often resulting in backtracking and potentially involving a considerable amount of search, since test generation is in the worst case NP-complete.

6.5.3 Cost Considerations

Underlying the preceding analysis are a number of assumptions about the relative costs of probing, test application, and computation, where the "cost" of an action is typically taken to mean the amount of time it takes to perform.

Analysis aimed at selection of optimal probes is useful only when computation is reasonably cheap compared to the probes themselves. There would, for example, be little point in waiting for a ten-minute computation to determine the optimal probe if all of the measurements are easily made in five minutes. In general the assumption holds true, partly because computation keeps getting cheaper and gets cheaper faster than almost anything else. Probing, by contrast, typically means some sort of physical action (hence is it likely to be slower), and some of those actions may result in losing information (e.g., having to move boards to get access to probe points). Hence the assumption is typically valid, but it is important to be explicit about it.

Similarly, generation of distinguishing tests is useful only when the required computation is cheaper than probing or when probing is impossible. As above, there is little point in waiting for a computation to construct an informative test if many measurements that would eliminate suspects could be made in the meantime. Although an adequate working assumption, it is violated occasionally because test generation can be expensive (NP-complete for combinational digital circuits).

Finally, an assumption underlying the preceding discussion is that probes are independent of one another and all have equal cost. This assumption is violated if there are a range of technologies for probing the device, each with its own cost, resolution, and number of resulting observations. A digital logic analyzer, for example, yields detailed observations of several signals simultaneously, but requires much more setup time than a simple voltmeter. Hence the voltmeter may be preferable to the logic analyzer even if it yields less information about the currently outstanding suspects.

Similarly, tests may have different setup costs — in fact they may have different setup costs depending upon the order in which they are applied — with analogous consequences. The potentially relevant literature on decision theory is too large to survey here, nevertheless it is important to be aware that subtleties of this kind are likely to arise in real applications.

7 INTERIM CONCLUSIONS

We have discussed a substantial collection of ideas and techniques that form the current basis for model-based diagnosis and troubleshooting. A brief review of the highlights will help set the stage for exploring the open research issues.

- Model-based troubleshooting is based on the comparison of observation and prediction.

Observation indicates what the device is actually doing, prediction describes the intended behavior. Discrepancies between the two provide the starting point for diagnosis. An important part of the diagnostic ability of model-based reasoning is provided by behavior descriptions that capture both the causal behavior of the device (predicting outputs from inputs) and inferences that can be made about its behavior (determining inputs from outputs).

One of the important consequences of the model-based view is the ability to view misbehavior as anything other than what the device is supposed to do. We need not pre-enumerate the kinds of things that might go wrong.

- Model-based troubleshooting is device independent.

Given a new device description, work can begin immediately on troubleshooting the new device. Unlike rule-based approaches, there is no time-consuming accumulation of experience. These systems reason instead from engineering principles applicable to a wide variety of devices.

- Model-based troubleshooting is symptom directed.

It reasons from the observed misbehavior toward the underlying fault. This is particularly important for any device complex enough that the set of correct behaviors is too large to explore exhaustively. In that case it is infeasible to run the device through all its correct behaviors to see which one is not working; we work instead from the information supplied by the symptom. The technique is also familiar, in the sense that it captures some of the intuitions and reasoning that experienced people typically use.

Model-based diagnosis can be understood as a process of hypothesis generation, testing and discrimination. Hypothesis generation works from a single symptom to determine which components might have caused that symptom. The key issue is providing a generator that is both complete and informed. We reviewed three different ways to do that, moving from the simplest version to more sophisticated approaches.

Where hypothesis generation works from a single symptom, the goal in hypothesis testing is to determine which candidates can account for all the observations available about the behavior of the device. We examined four approaches, ranging from straightforward fault simulation, to constraint suspension, DART's use of resolution residue, and the GDE approach.

In hypothesis discrimination the fundamental issue is finding inexpensive ways to gather additional information that will distinguish among the surviving hypotheses. In exploring probing strategies we looked at four ideas that used successively more information, beginning with structure, adding information about behavior, *a priori* failure probabilities, and finally ending with a means of estimating which probe will likely yield the shortest sequence of measurements. A brief review of test generation demonstrated that the traditional technique is indiscriminate in its selection of components to use in constructing a test; considerable advantage can therefore be gained by the simple expedient of using only known good components.

Two important elements of the analysis in this paper are the view of the basic task as a three step process of generate, test, and discriminate, and the exploration of the character and amount of knowledge that can be brought to bear at each step. Dividing the task into those three steps provides an important form of mental hygiene, making it possible to understand each of these fundamentally different problems on its own terms, without being misled by the common implementation practice of intermingling them for efficiency. Exploring the kinds of knowledge used at each stage offers a sound basis for comparing different variations and understanding how and why one may be more powerful than another.

The combination of these two elements also maps out a sizable space of program architectures. This is valuable because it provides a way of unifying what might otherwise appear to be a diverse collection of systems. We claim in fact that the model-based systems built to date fit comfortably somewhere in that space, i.e., all the current systems can be characterized in this framework according to the amount and kind of knowledge they use at each stage.

One overall consequence evident at this stage is that model-based diagnosis is a fairly well-understood process. Part of the evidence for this is the character of the different programs that have been built: The variations in the way they work are minor in comparison with the common core of techniques in use. Additional evidence comes from recent success at recasting much of the reasoning in terms of formal logic. The work in [Reiter87] and [Ginsberg86], for instance, provides formal definitions of

and proofs for some of the ideas presented in more intuitive form here.

All this has two interesting consequences. First, the technology is ready for application. A body of understanding is in place that is sufficient to attack modest-sized but real problems. Building these applications will no doubt raise additional interesting questions, but there is a sufficient base of knowledge available for us to begin to use it.

Second, the technology is well enough understood that the interesting research agenda now consists of either developing substantial advances beyond the techniques outlined earlier or finding fundamentally different ways to proceed. Interesting applications may result from constructing, tracing, and reasoning about dependencies, but research contributions arise by exploring problems for which the existing techniques are inadequate and finding ways to make substantial advances in them.

We consider next a number of problems that may help spur such results.

8 THE RESEARCH ISSUES

Three categories of research issues seem particularly important and promising at this point in the evolution of the art: Device independence and domain independence, scaling up to more complex behaviors, and selecting the "right" model. The first addresses the question of how broadly we can use the current set of ideas. The case for device independence is easily made, since nothing done so far is specific to the particular device(s) examined, but are the ideas more broadly applicable? What happens if they are applied to devices built with entirely different technologies?

Numerous questions arise in considering scaling up to more complex behaviors. At the most basic level, the question is how to represent and reason about the behavior of more complicated devices, in particular those that have memory and thus can present interesting dynamic behavior. A related question is the power of our predictive engines: How can we improve their performance so that predictions can still be made when dealing with complex devices or complex interaction topologies?

Finally, the question of selecting among models confronts a number of very difficult problems. As will become clear, the difficulties start with acknowledging the apparently simple observation that model-based reasoning is only as good as the models we provide to it. That will lead to an interesting and difficult challenge — the battle between complexity and completeness, where the desire to be complete in diagnosis seems directly contradicted by the impossibility of dealing with an unconstrained problem.

8.1 DEVICE INDEPENDENCE AND DOMAIN INDEPENDENCE

It appears easy to argue that the technology reviewed so far has a strong degree of device independence — given a new description of a different circuit, the same

reasoning process can begin immediately. It is not so obvious, by contrast, what degree of domain independence these techniques exhibit. While there has been a small amount of work done in other domains (e.g., neurophysiology, hydraulic systems), the vast majority has been aimed at relatively simple electronic circuits.

At this point an intriguing experiment would be to go out on the edge and apply this in a domain where it is not at all obvious that it will work. It would, for instance, be fun to try thinking about clock repair in this fashion. Not the modern digital kind, though; the interesting challenge would be the old-fashioned gear, wire, and spring-driven models. What would it take to describe the behavior and structure of such a device? Can the techniques reviewed above be used to reason about it? The intent here is to work on a problem that strains the state of the art, to teach us more about representing and reasoning about structure and behavior.

8.2 SCALING

In considering whether and how this technology can scale up to larger devices, it is important to recognize that there are at least two independent dimensions — size and complexity — and that size alone is not a particularly difficult issue. If the basic components are simple, it is possible to work with thousands of them without straining the current technology. One current program, for instance, models and diagnoses a system with a few thousand components [First82]. Each of them is very simple, but nothing new is required to apply the existing ideas to this system of thousands of parts. The model entry task may be sizable, but it is an engineering challenge, not a fundamental advance in representation or reasoning.

More interesting challenges arise when we start to deal with devices with complex behavior. As one commonsense example, consider the behavior of a VCR that can be programmed to record two different broadcasts at different times in the future. Even this relatively modest-sized finite state machine can present apparently daunting problems of representing and reasoning about behavior.

As a somewhat more immediately useful example, consider the task of describing the behavior of an ALU (arithmetic/logical unit), using the behavior representation technology available today. If that seems tractable, imagine describing the behavior of a common microprocessor like the 80386. How might we describe what that device can do in a way that makes possible examining and reasoning about it? As long as we're at it, imagine describing the behavior of something genuinely complex, like a disk controller.

Nor is complexity solely the province of large-scale devices. Work at the other end of the scale has demonstrated how complex the behavior of a single transistor can be when coarse abstractions like "switch" or "amplifier" prove to be insufficiently detailed [Dague87]. Many of the same issues arise here as well.

What might be done? One approach is to look for a new vocabulary, a new set

of abstractions designed to deal with the kinds of complexity encountered. Imagine examining the data sheet for the 80386, making careful note of the vocabulary in use. That data sheet is a form of existence proof: With some degree of success it communicates what this device is supposed to do. The easy speculation is that its success arises in large part because it uses the "right" set of abstractions. The more difficult part is understanding what "right" means — what makes these abstractions effective? What is it that they ignore, what do they emphasize, and why are those effective selections?

Complex behavior also forces the question of the adequacy of our predictive engines. As noted earlier, the simpler local constraint propagators stall when encountering the need to deal with more than one equation in one unknown. Although some effort has been directed toward propagating symbolic expressions, the resulting algebra can be quite complex. One possible approach to the problem would be to guide the algebraic manipulations with some knowledge of the device structure and behavior, similar in spirit to the observation that a physicist guides his mathematics by an understanding of the problem and what he is trying to establish. The question is not how to be good at symbolic manipulation of complex expressions, so much as it is knowing what symbolic manipulation to do to avoid the complexity in the first place.

A third set of challenges arises in dealing with devices with memory. If, as is frequently the case with such devices, we know only the inputs supplied to it initially and the final output that results some time later, hypothesis generation and testing becomes truly indiscriminate. Work reported in [Hamscher84], for instance, examined the task of diagnosing a sequential multiplier (a device that multiplies one digit at a time, shifting and adding in much the same way the problem is done by hand). If the multiplier's behavior is modeled using the technology reviewed above, candidate generation becomes indiscriminate — almost every component can account for the misbehavior. This is not a minor consequence of current implementations; the difficulty arises from the basic nature of the problem: If all we know is the input at the beginning and the output at the end, the problem is genuinely underconstrained in much the same way that two equations are insufficient to determine the value of three unknowns.

This is a second place in which new abstractions may prove to be the relevant tool, particularly temporal abstractions. Some early work in this direction has been done and seems promising: [Hamscher88], for instance, demonstrates how temporal abstractions can be effective for such devices.

One other approach that may prove effective in reasoning about complex devices is the notion of "second principles of misbehavior." One example is the heuristic that, in a complicated device fault manifestations will be obvious. To illustrate, imagine working with a device that includes a current generation microprocessor, one

that happens to be broken in some fashion, and consider the consequences of that fault on the microprocessor's behavior. It is possible, but highly unlikely that the consequences will be subtle: It is unlikely for instance that the device will exhibit only a very small perturbation in its expected behavior for only one of the instructions in its instruction set. It is much more likely that the fault will result in some obviously aberrant behavior every time the device is used. One common form of that aberration is for the device to stop producing any behavior at all, e.g., by hitting an illegal instruction and halting.

This is one example of the second principle that complicated devices don't break in subtle ways. It is a "principle" in the sense that it can be explained by (and perhaps eventually derived from) arguments about design. In this case, for instance, the argument is that complex designs often involve re-use of modules, both to simplify the design and reduce cost. Re-use of modules in turn means that any error in such a module will tend to have widespread consequences. In a microprocessor, for instance, a single ALU may be used both for the arithmetic required for an ADD instruction and the arithmetic needed to compute the next instruction address. Any error in that ALU will not only yield incorrect sums (which might be overlooked), it will also introduce instruction sequencing errors that are unlikely to be missed.

Since these principles can be grounded in knowledge about design, they are more than device-specific heuristics and are likely to have widespread applicability. They are also an important addition to the ideas explored thusfar, because we are as a field a long way from being able to do such reasoning from a purely first principles approach. Second principles of misbehavior thus offer a way of summarizing what would otherwise be a long and difficult derivation.

One challenge we face is finding more of them; one obvious place to start is with experienced troubleshooters. Whenever a model-based system produces a diagnosis that is logical but strikes a human troubleshooter as inappropriate, there is the standard opportunity to find out what it is that the experienced troubleshooter knows that is still missing from the system. Some of that knowledge may point toward additional second principles of useful breadth and utility.

8.3 MODELING IS THE HARD PART

The third and possibly most intriguing area of research is brought into focus by acknowledging that all model-based reasoning is only as good as the model. This observation is in some ways obvious and in some ways fairly subtle, but the consequences are interesting and present difficult problems.

To illustrate one version of the problem, note that all of the reasoning techniques reviewed earlier generate predictions by propagating along the pathways shown in the device description, then traced back from the discrepancies along those same pathways to find suspects. The crucial point is twofold: Suspects are found by tracing causal

pathways back from a symptom, and all of the reasoning above accepted the device description as given, implicitly assuming that the pathways supplied accurately model causality in the device. Yet this can easily be false.

One commonplace example of this phenomenon is a bridging fault, the event that results when a chip is being soldered in place and enough solder accumulates at two adjacent pins to bridge the gap between them (Figure 21). The result is a connection — a causal pathway — where none was intended.

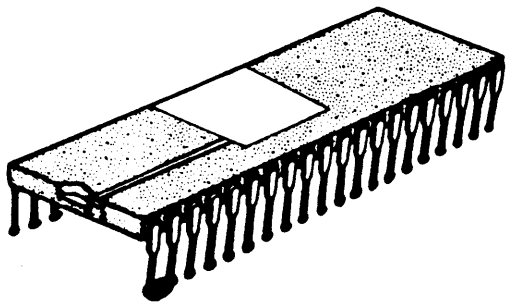


Figure 21: A Solder Bridge.

The possibility of faults of this sort has a particularly interesting consequence. Since candidates are found by tracing back along causal pathways, if the pathways indicated by the device description are different from those in the actual device, the tracing process will lead to the wrong components. Put somewhat more simply, the great virtue of the model-based approach is its ability to reason from the description of structure and behavior, yet the fatal flaw in the model-based approach is that it reasons from the description of structure and behavior, *and that description might not capture the actual causality in the device.*

8.3.1 The Model Must Be Wrong

How is it that the model might not be an accurate description of the causality in the device? One possibility is that the device isn't supposed to be that way. The

bridge fault is one example of this, another is an error during assembly — the device is simply wired up incorrectly.

A second possibility is unexpected pathways of interaction. In an electronic circuit, a wire is the expected pathway of interaction; that's how components are supposed to affect one another. But there can be other, unexpected, pathways as well: One component may heat up another, two wires carrying high frequency signals may be so close that they affect one another via electromagnetic radiation, etc. The important point is that the design description, by *intent*, only tells us about the pathways of interaction that are supposed to occur. In the device itself other unknown pathways may be operating. The consequences of this are particularly evident in DART's explicit statement that its diagnosis is restricted only to "...propositions from the design description or their negation." Hence the only kinds of diagnoses it can even consider are those stating that some component explicitly mentioned in the design description is malfunctioning.

Third, the model may not match the device because in our routine practice we explicitly decide not to represent a particular level of detail. In a large circuit, for instance, we may choose not to model every individual wire, settling instead for a slightly more coarse-grained model in which components are modeled as connected directly to one another.

But most importantly, it is in principle necessarily true that the model be different from the device. It is the fundamental nature of all models, all representations, that is at issue here: There is no such thing as an assumption-free representation. Every model, every representation contains simplifying assumptions. That's what models are, so in some ways this is perfectly obvious.

The perhaps less obvious part is the unavoidable impact this has on model-based reasoning. As noted, the fundamental idea behind the technique is the idea that, if the model is correct, then all the discrepancies between observation and prediction arise from, and can be traced back along causal pathways to, defects in the device. But the model is, inevitably and in principle, *never* correct.

To be more precise, the model is never *completely* correct. When it is a good enough approximation, the techniques described earlier are successful. But the inevitability of incorrectness in theory and the pragmatic reality of it in practice mean that this issue is real and crucial to the robustness of the systems we build. We need to understand both what effect it has on the systems we build and how to deal with it.

8.3.2 Consequence: Complexity vs. Completeness

One of the most important effects of the phenomenon is an inevitable tension between complexity and completeness. To be complete, diagnostic reasoning would have to consider all the things that may possibly go wrong, along every *possible* pathway of

interaction. But such reasoning would be indiscriminate, implicating every component — there would always be some (perhaps convoluted) pathway by which that component might have caused the problem. Yet if we make any simplifying assumptions, i.e., omit any pathway, there will be entire classes of faults that the system will never be able to diagnose.

There is a fundamental problem here. If we make any simplifying assumptions we run the risk of being incomplete, because the simplifying assumption might be the one that encompasses the actual fault. Yet without some simplifying assumptions the reasoning drowns in complexity.

While this arises in a particularly immediate fashion here, it appears to be a fundamental issue for problem solving in general. Any time we set out to solve a problem, we need to make simplifying assumptions about the world in order to get started, yet sometimes those assumptions are wrong. Thus any techniques that can help us to select, organize and manage the assumptions that will be of potentially broad utility.

8.3.3 Consequence: Model Selection Is Fundamental

Perhaps the most interesting implication of this line of argument is the significance of the problem of model selection. Since there are no assumption-free representations, one strategy would be to assemble a collection of them, each embodying a different set of assumptions, along with a body of knowledge about how to select carefully from among them. No one of them or any simple combination of them provides a complete representation, but progress might be made by selecting carefully from among them, attempting to make enough assumptions to keep the problem tractable, yet making a few as possible to reduce the chance of not being able to see the actual problem.

It is likely as well that the choice will not only have to be judicious, but repeated and dynamic as well, changing views on the fly as understanding of the problem evolves. One support for this approach is the observation that experienced engineers do something like this. We need to understand what it is they know and how they reason about model selection.

The problem seems to lie at the heart of engineering problem solving: Perhaps the most basic, most important decision made in starting to solve a problem is deciding "how to think about it." What is it that suggests modeling something as an analog device, a digital device, or a hydraulic device? How do we know what's relevant? How does the process begin? The problem seems difficult but particularly intriguing.

Three speculations suggest possible approaches to the problem. First, we might review the difficulties mentioned above that are encountered when using models, and reformulate them as heuristics for model design [Hamscher88]. The difficulty presented by reconvergent fanout (i.e., causing local propagation to stall) can, for instance, be reduced to some degree by selecting module boundaries to encapsulate

the fanout. Similarly, judicious selection of module boundaries can help reduce hidden state, the problem that makes diagnosis underconstrained in the case of the sequential multiplier. A set of such heuristics would assist in the design of models that reduce or avoid some of the problems encountered above.

A second speculation explores the problem of deciding how to model something by suggesting that different pathways of interaction define different kinds of models, different representations, which can then be layered to provide a sequence of successively more complex views [Davis84]. A wire, for instance, is one pathway of interaction, it defines the traditional schematic. If heat is the relevant pathway, that defines a different representation of the device, one in which "distance" is defined in terms of how easily one device heats another. Electromagnetic radiation is a third pathway that defines yet another kind of model and another distance metric.

These multiple different kinds of models are then organized from simplest to more complex (defining "simplicity" is itself an open issue), so that the system starts by using the simplest and falls back on more complex models only as necessary. The technique has been used to diagnose a bridging fault successfully, demonstrating that multiple models using different representations and different definitions of distance can be used to reduce complexity without permanently losing completeness [Davis84].

A third speculation begins with the observation that every model is defined by a set of simplifying assumptions. We might collect the set of all the simplifying assumptions routinely made and consider the space of models that are generated by it. For example, Figure 22 shows three different models of a NAND gate, beginning with the traditional transistor level model at the bottom.

Assuming that power can be ignored, then abstracting away from the specific subcomponents to the roles they play, produces the intermediate level representation in the middle. Two further simplifying assumptions — that current can be ignored and that all the subcomponents can be encompassed by a single box, yield the traditional representation at the top. Hence these two pairs of assumptions yield two successively simpler models of the device.

But these are not the only models those assumptions can generate. The simple trick of changing the order in which the assumptions are made produces an entire lattice of different models (Figure 23).

Some of them are admittedly rather obscure, but there are in fact (perhaps obscure) circumstances under which every one of them will be the "right" way to think about the device. One reason why some faults are so difficult to diagnose may be precisely because the "right" model in that case is a particularly unusual set of assumptions. Even faults as commonplace as bridges illustrate the issue: Part of the reason they are especially difficult to diagnose is that they require examining a less familiar representation — the physical layout of the chips. While the fault is "simple" in that representation (two adjacent pins), it can appear on the functional diagram

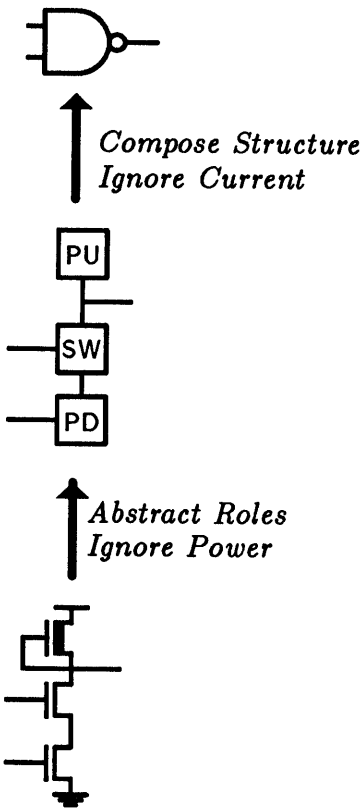


Figure 22: A Simple Hierarchy of Models.

as a connection between two widely separated points.

This is, of course, still speculation. Given that the lattice in Figure 23 was generated simply by changing the order of the assumptions, there's no particularly compelling reason to believe that it will work well. Nor have we answered the second half of the question: How to select from among the models, and how to know which to choose next when one of them begins to fail. This is only a beginning, but it may be worth further consideration.

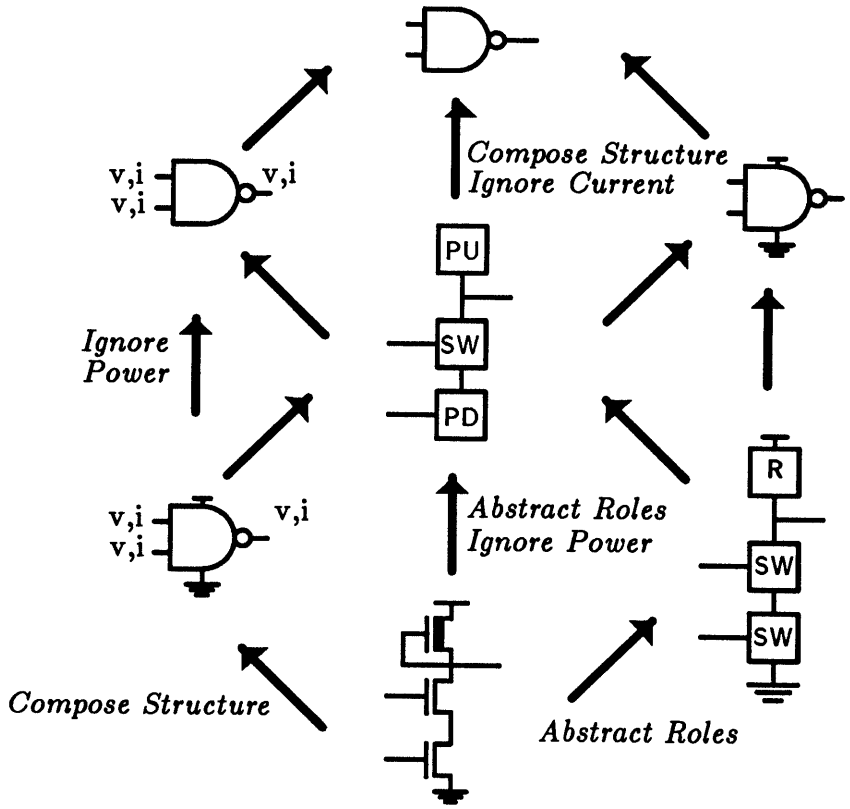


Figure 23: A More Complex Hierarchy of Models.

9 SUMMARY

We began this survey by viewing model-based diagnostic reasoning as the interaction of prediction and observation, and saw that one useful consequence was the chance to view misbehavior as anything other than what the device is supposed to do. Model-based reasoning thus covers a broader collection of faults than traditional approaches to diagnosis. A second virtue of the technique is its device independence, enabling us to begin reasoning about a system as soon as its structure and behavior description is available.

In examining how to represent structure, we noted the utility of descriptions that were hierarchic, object-centered, and topologically identical to the device being modeled. In examining behavior we noted the widespread use of constraint-like descriptions that allow both simulating the actual behavior of the device and making inferences about what the values at inputs must have been.

We explored diagnostic reasoning by viewing it in the three phases of hypothesis generation, test, and discrimination. This view allowed exploration of each of these fundamentally different problems on its own terms, made clear the common core of techniques that are in use, and offered evidence for the claim that model-based systems to date fit into the space of architectures characterized by the amount and kind of knowledge they use at each stage. The view also supports the claim that the process is reasonably well understood: Building a dependency-tracing model-based reasoner is now a fairly routine process.

Finally, we examined three major open research issues. We explored the question of domain independence, leading to the suggestion of trying these techniques on devices from widely different domains, to extend our ability to describe structure and behavior. We examined the difficulties in scaling up to devices with considerably more complex behavior, speculated about the possibility of finding a new vocabulary of effective abstractions, and touched on the difficulty of producing predictions in the face of complex behavior. And we emphasized the fundamental role and fundamental difficulty of model selection as the central problem in both extending the reach of these programs and ensuring their robustness.

Acknowledgments

Many people contributed useful suggestions aiding both the research and writing of this paper, including Johan deKleer, Mike Genesereth, Paul Resnick, Mark Shirley, Howie Shrobe, Reid Simmons, Jeff Van Baalen, Dan Weld, Brian Williams and Peng Wu.

REFERENCES

[Brown76]
Brown, A. L. *Qualitative Knowledge, Causal Reasoning, and the Localization of Failures.* Technical Report AI-TR-362, MIT Artificial Intelligence Laboratory, 1976.

[Brown82]
Brown, J. S., R. R. Burton, and J. de Kleer. Pedagogical, Natural Language, and Knowledge Engineering Techniques in SOPHIE I, II, and III, in: D. Sleeman and J.S. Brown (Eds.), *Intelligent Tutoring Systems*, (Academic Press, New York, 1982) 227–282.

[Dague87]
Dague, P., O. Raiman, and P. Deves. Troubleshooting: When Modeling is the Difficulty, In *Proceedings of AAAI-87*, Seattle, WA, pp 600–605, August 1987.

[Davis82]
Davis R., H. Shrobe, W. Hamscher, K. Wieckert, M. Shirley, and S. Polit. Diagnosis Based on Structure and Function, In *Proceedings of AAAI-82*, Pittsburgh, PA, pp 137–142, August 1982.

[Davis84]
Davis, R. Diagnostic Reasoning Based on Structure and Behavior. *Artificial Intelligence* 24(3):347–410, December, 1984.

[deKleer76]
de Kleer, J. *Local Methods for Localizing Faults in Electronic Circuits.* Memo 394 (out of print), MIT Artificial Intelligence Laboratory, 1976.

[deKleer87]
de Kleer, J., and B. C. Williams. Diagnosing Multiple Faults. *Artificial Intelligence* 32(1):97–130, April 1987.

[First82]
First, M. B., B. J. Weimer, S. McLinden, and R. A. Miller. LOCALIZE: Computer-Assisted Localization of Peripheral Nervous System Lesions. *Computers and Biomedical Research* 15(6):525–543, December, 1982.

[Genesereth84]
Genesereth, M. R. The Use of Design Descriptions in Automated Diagnosis. *Artificial Intelligence* 24(3):411–436, December 1984.

[Ginsberg86]
Ginsberg, M. Counterfactuals. *Artificial Intelligence* 30(1):35–80, October 1986.

[Hamscher84]
Hamscher, W., and R. Davis. Candidate Generation for Devices with State: An Inherently Underconstrained Problem. In *Proceedings of AAAI-84*, Austin, TX, pages 142–147. AAAI, August, 1984.

[Hamscher87]
Hamscher, W. and R. Davis. Issues in Model Based Troubleshooting. MIT AI Lab Memo 893, March 1987.

[Hamscher88]
Hamscher, W. Representations for Model Based Troubleshooting. March 1988, available from the author.

[Pan84]
Pan, J. Qualitative Reasoning with Deep-level Mechanism Models for Diagnoses of Mechanism Failures. In *Proceedings of CAIA-84*, Denver, Colorado, pages 295–301. IEEE, December 1984.

[Patil81]
Patil, R., Szolovits, P., Schwartz, W. Causal understanding of patient illness in medical diagnosis in *Proceedings of IJCAI-81*, Vancouver, BC, pp 893–899, August 1981.

[Reiter87]
Reiter, R. A Theory of Diagnosis from First Principles. *Artificial Intelligence* 32(1):57–96, April 1987.

[Scarl85]
Scarl, E., J. R. Jamieson, and C. I. Delaune. A Fault Detection and Isolation Method Applied to Liquid Oxygen Loading for the Space Shuttle. In *Proceedings of IJCAI-85*, Los Angeles, CA, pages 414–416. IJCAI, August 1985.

[Shirley83]
Shirley, M. H., and Randall Davis. Generating Distinguishing Tests based on Hierarchical Models and Symptom Information. In *IEEE International Conference on Computer Design*, 1983.

[Sussman80]
Sussman, G. J., and G. L. Steele. Constraints: A Language for Expressing Almost-Hierarchical Descriptions. *Artificial Intelligence* 14(1):1–40, January, 1980.