Massachusetts Institute of Technology
Artificial Intelligence Laboratory

A.I. Memo 893

March, 1987

Issues in Model Based Troubleshooting

Walter Hamscher
Randall Davis

Abstract. To determine why something has stopped working, it's helpful to know how it was supposed to work in the first place. This simple fact underlies recent work on a number of systems that do diagnosis from knowledge about the internal structure and behavior of components of the malfunctioning device. Recently much work has been done in this vein in many domains with an apparent diversity of techniques. But the variety of domains and the variety of computational mechanisms used to implement these systems tend to obscure two important facts. First, existing programs have similar mechanisms for generating and testing fault hypotheses. Second, most of these systems have similar built-in assumptions about both the devices being diagnosed and their failure modes; these assumptions in turn limit the generality of the programs. The purpose of this paper is to identify the problems and non-problems in model based troubleshooting. The non-problems are in generating and testing fault hypotheses about misbehaving components in simple static devices; a small core of largely equivalent techniques covers the apparent profusion of existing approaches. The problems occur with devices that aren't static, aren't simple, and whose components fail in ways current programs don't hypothesize and hence can't diagnose.

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

1 Introduction

Programs for doing automated diagnosis from structure and behavior strive for generality of various kinds. One aspiration is to have programs able to diagnose virtually any designed artifact in a particular technology. A more ambitious generality is implied by the dream of building a general troubleshooter that could diagnose (say) automobiles as well as analog circuits, simply by substituting different types of components for each domain.

Our claim is that the dream is both closer and farther away than is commonly appreciated. It is closer, because most existing programs use similar techniques and the commonality suggests that a "domain-independent" troubleshooting methodology is within reach. It is farther away, because these same programs have built-in assumptions about their domains which must be made explicit before they can be generalized. The difficult issues in this line of research do not arise in the methods themselves, but rather from the simplifying assumptions implicitly built into them.

A number of programs reason from structure and function to diagnose devices in a variety of domains, using what appears to be a variety of techniques, including INTER [deKleer76], WATSON [Brown76], SOPHIE [Brown82], LOCALIZE [First82], [Davis84]'s program, DART [Genesereth84], IDS [Pan84], LOX [Scarl85], and the ATMS troubleshooter [deKleer87].

The variety of domains and computational mechanisms found in these programs tends to obscure important similarities. One set of similarities concerns troubleshooting techniques. These similarities can be made clear by describing them in terms of the generate-and-test paradigm, illustrating the ways different programs use the same kinds of knowledge.

A second important set of similarities concerns the assumptions that different programs make about the kinds of components and faults to be encountered. Among these assumptions are that components have no hidden state and that the given representation of interactions between components is complete and correct. These assumptions are often built into programs for the sake of efficiency, resulting in important limitations. These limitations in turn constitute an agenda of open problems in automated diagnosis.

2 Diagnosis from Structure and Behavior

Given some observations of a misbehaving device, a description of its internal structure, and descriptions of the behavior of its components, we wish to find out which components could have failed in such a way as to explain the misbehavior. A useful way to decompose this task is to consider three separate tasks: (i) generating fault hypotheses, (ii) checking those hypotheses for consistency, and (iii) discriminating among the consistent hypotheses on the basis of further probes or tests. This section discusses each in turn.

It is necessary to make some initial definitions and assumptions, each of which will be reexamined later.

A **component** is a part of a **device**. Diagnosis programs diagnose devices to find faulty components. **System** is used interchangeably with "device" to refer to a larger collection of components, such as a computer system.

The **structure** of a device can be thought of as a graph, with the components represented as nodes and connections between components represented as arcs. **Terminal** is used to mean a point where a component can be connected to others.

A **suspect** is a component whose misbehavior could possibly explain one or more symptoms. For the moment, let a **fault hypothesis** be a specific misbehavior hypothesized for a suspect.

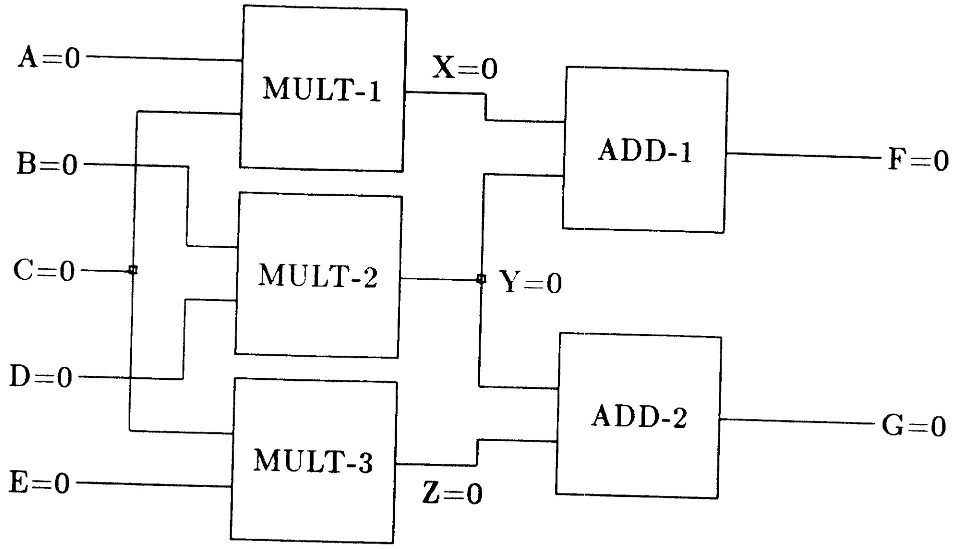
As an example, the structure of a digital device might be represented with the logic chips as "components," the wires as "connections," and the the pins on the chips as "terminals." A different representation of the same device might have the components represent boolean logic gates, the connections represent electrical connectivity through metal wires and pins, and the terminals represent the signal inputs and outputs of the gates.

From the point of view of a diagnosis program, these are two equally valid representations of "structure" for the same device. Suspects generated from the two representations will be different, because the components and their connections are different, but the diagnosis methods to be discussed are flexible enough to deal with these and other notions of "structure."

2.1 Hypothesis Generation

The generate-and-test paradigm requires that the generator of candidate solutions be complete, in the sense that every potentially valid solution will eventually be proposed. Given a device description, a complete generator of fault hypotheses could be trivially built by exhaustively enumerating all components, since all suspects are components. But not all components are valid suspects; suspects should explain the observed symptoms without implying symptoms that were not observed. It is advantageous to incorporate this constraint into the generator, so that fewer invalid suspects are proposed. There exist a number of progressively more elaborate ways to use knowledge about the device's structure and its components' behavior to generate a more constrained set of hypotheses while preserving the required property of completeness. In this section we begin with an extremely simple version of hypothesis generation and develop these elaborations one at a time.

A **discrepancy** is a disagreement between an observation of a device's behavior and its expected fault-free behavior. For example, the adder-multiplier circuit shown below presented with zeroes on all inputs is expected to produce a zero on output F. An observation of anything else at that terminal constitutes a discrepancy. A program's first task is to determine whether any discrepancies exist. This can be done by simulating the device's expected behavior given the inputs presented and comparing the results to observations of the real device.



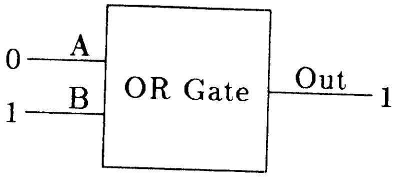
Adder-Multiplier Example

Given these discrepancies, a simple, intuitively appealing way to find suspects is to find all the components connected to a discrepancy via some path through the connections. This makes sense because the suspect must be among the components that could influence the expected value, and according to the model this influence could only be exerted through the connections. Suspicion is "contagious," in the sense that a discrepancy observed at one of the terminals on a component implies either that the component is malfunctioning, or that the component is normal but some component connected to it is malfunctioning. Each of the other terminals of the component yield further suspects and further discrepancies, etc. The problem with this approach, however, is that in most cases following every connection means that *every* component will be reached, so this is a poor strategy.

Intuition says that a better approach would be to identify the direction of causality in the device, and mark as suspects only those components that are "upstream" of discrepancies. In the example above, a discrepancy observed at output F would make suspects of only the three components upstream from F. Knowledge about components' direction of operation can thus constrain the suspects generated. Components that have identifiable input terminals and output terminals are said to have **directionality**. This notion is not applicable in every domain; analog electronic components such as resistors, for example, are not usually thought of as having inputs and outputs. Nevertheless, the technique is appropriate in many domains, so we will pursue some of its elaborations.[1]

One way to elaborate is with a **behavior model**. This is information about a component that can be used to predict its response given its inputs. This can then be used to constrain hypothesis generation: when a discrepancy is observed at an output, we need move upstream only from those inputs upon which the expected output depended.

For example, suppose a digital OR gate is expected to get a 0 on input "A" and a 1 on input "B," yielding a 1 on the output. The output of 1 in this situation depends only on B's being 1. Hence, if the output is observed to be zero, only B need be traced upstream.



OR gate

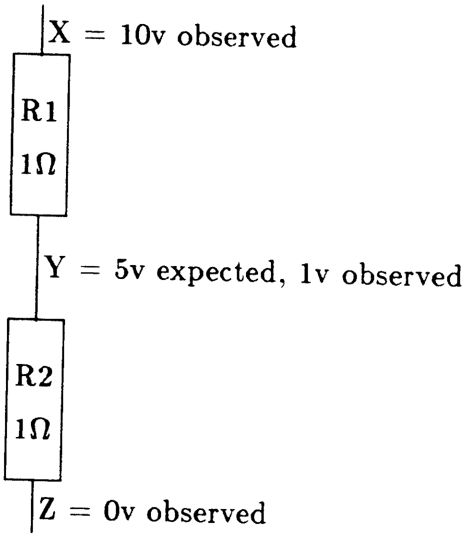
There are different ways to implement this approach to generating suspects. One method is to record dependencies whenever an output prediction is made. In the OR-gate example, the relevant dependency would be created and stored with the original deduction that the output should have been 1. *Dependency based* hypothesis generation schemes follow these dependency records upstream from discrepancies. Each component visited while tracing these dependencies back to primary inputs is a suspect. Davis' program stores explicit dependencies for this purpose. Another method traces inputs upstream by computing the logical consequences of observations. In this example, we know that "*if* an OR gate is normal and one of its inputs is 1, *then* its output is 1." Since this output is observed to be something other than 1, then either the OR gate is not normal, or neither input is 1. Hence, either the OR gate is not normal, or the B input was not 1. Hence, either the OR gate is not normal, or the component upstream of B is not normal, etc. DART exemplifies this *inference based* method of computing suspects; LOX takes a similar viewpoint. These various implementations of upstream tracing yield identical suspects.

[1] The technique is also worth studying because it is well grounded in intuition, some of the programs being surveyed assume it, and it can be generalized easily.

The notion of *conflicting assumptions* provides a more general framework than the intuitive notion of upstream tracing. In this view, each discrepancy represents a conflict between expectations and observations. Assumptions about the correct behavior of components are recorded at simulation time and underly those expectations. The existence of a discrepancy means that the set of assumptions is inconsistent and hence at least one underlying assumption must be false, i.e. at least one of the components assumed to be behaving correctly is actually misbehaving.

In domains for which components' causal direction is the sole source of dependencies, there is little distinction between the "upstream tracing" and "conflict-oriented" views. The discrepancy at F in the adder-multiplier example yields the suspects ADD-1, MULT-1, and MULT-2 under both approaches. However, the conflict-oriented view facilitates dealing with components having no directionality, since it requires no distinction between inputs and outputs, neither for the individual components nor the device as a whole. So long as all values are predicted with all the relevant assumptions being recorded, the technique will generate a complete set of suspects.

The figure below shows a trivial circuit with two resistors. Suppose the potentials at nodes X and Z are known to be 10 volts and 0, respectively. The voltage at node Y is measured to be 1 volt, instead of 5 as expected. This is a discrepancy, or more accurately, it is a conflict between two assumptions, namely, that R1 and R2 both have resistances of 1 ohm. At least one of these assumptions must be false, hence both resistors are suspects.



Voltage Divider Example

This simple example also illustrates a characteristic of devices composed of nondirectional components, which is that any single prediction may depend on a large portion of the components working properly. As a result, hypothesis generation will be unavoidably indiscriminate.

Given this variety of hypothesis generation techniques, the proper method to use in a program can be suggested in part by the class of devices the program is expected to diagnose. We have seen three techniques of increasing generality:

1. Upstream tracing is adequate in domains with simple, directional components. LOCALIZE is able to use this technique thanks to the trivial behavior of neural pathways in its domain. IDS uses it as well, in a representation that shows only the intended direction of information flow between components in an otherwise nondirectional device, thereby risking an incomplete generator.

2. Various mechanisms can constrain upstream tracing by using components' behaviors. DART, Davis' program, and LOX work in domains with moderately complex yet largely directional component behaviors, thus motivating the use of dependency-based and inference-based schemes.

3. Hypothesis generation can be broadly viewed as the task of finding conflicting assumptions. The domain of analog electronic circuits involves mainly nondirectional components, hence INTER, WATSON, SOPHIE take this conflict-oriented view, as does INTER's descendant, the ATMS troubleshooter.

2.2 Hypothesis Checking

Usually there are initial suspects that are locally consistent, but globally inconsistent. A suspect can be globally inconsistent either because it cannot explain observed discrepancies or because its misbehavior would imply discrepancies that were not observed. The purpose of hypothesis checking is to eliminate inconsistent suspects using only the observations at hand, i.e. without performing any further tests or internal probes of the device. As with hypothesis generation, there are progressively more elaborate and powerful ways to use such observations. As before, let us begin with a simple technique for hypothesis checking and develop more powerful elaborations of it one at a time.

One way to exonerate components is by using **corroborations** [deKleer76] – observations that agree with expectations. Intuition tells us that if an output of a component is normal, the component is functioning correctly and its inputs are normal. If those inputs were normal, then its immediate predecessors are functioning correctly, etc. This intuition is rarely correct, however. It assumes that (i) the input of a normal component can be determined solely from its output, that (ii) components only fail in such a way that misbehavior is detectable for every possible input. Rarely are components so simple in their behavior that this method suffices; LOCALIZE's domain of neural pathways is an exception.

A more powerful method for using corroborations to detect inconsistencies is **fault envisionment**: insert a hypothesized misbehavior and simulate to see whether it matches all observations (both discrepancies and corroborations). Note that this requires a predefined set of possible misbehaviors for each component type. For example, a resistor in an electrical circuit may be faulted by being "shorted"; the resulting misbehavior is that its two terminals are forced to have the same voltage. Any disagreement between the observed and predicted values rules out a hypothesis, and suspects are exonerated by ruling out all their possible misbehaviors.

An advantage of generalizing the notion of a "behavior model" to include the behavior of components when faulted is that **dependent failures** – failures that occur when a failure in one component damages other components – can be hypothesized and their effects predicted through fault envisionment [Pan84].

A disadvantage of fault envisionment is that the number of ways components in the domain can fail grows quickly with their physical complexity. In IDS [Pan84], for example, analog electronic components such as resistors and diodes can be assumed to fail only by having shorts or open circuits between two or more terminals. This works fine for components with 2 terminals, but becomes unwieldy when non-primitive components or primitive components with more terminals are considered: an n-terminal component will have at least $2(2^n - n - 1)$ such failure modes.

A more general approach than either of the preceding relies on the observation that a consistent hypothesis must account for all discrepancies. If there is more than one discrepancy, and only a single failure is assumed, the set of consistent suspects can be computed simply by the intersection of the suspect sets that arise from each discrepancy. Moreover, in addition to accounting for all discrepancies, a consistent suspect must also account for all corroborations, i.e. there must exist an assignment of values to its terminals such that all, and only, the observed discrepancies are produced. **Constraint suspension** [Davis84] and similar techniques do this by, in effect, attempting to infer what each suspects' misbehavior would be if it were indeed failing. The technique follows from the observation that the normal behavior of a component imposes a constraint on the values at its terminals. If the component is working correctly then that constraint is in force; otherwise the constraint is *suspended*, we simply don't know the relation between the component's terminals.

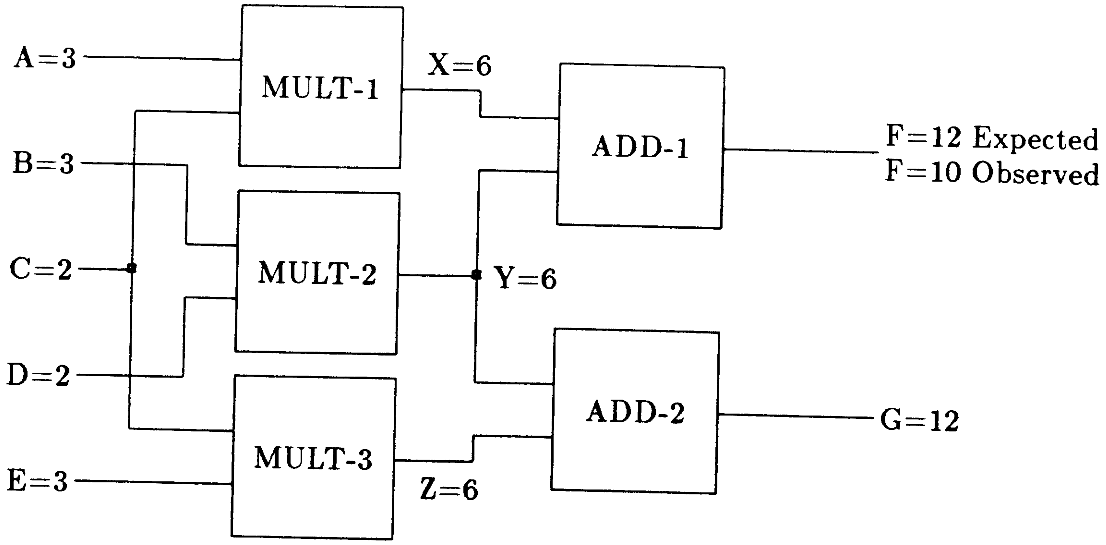
Consider, for example, an adder, whose behavior can be captured in terms of three rules: its output is the sum of its inputs; its first input is the difference of its output and second input; and symmetrically its second input is the difference of its output and first input. The latter two rules capture what we can infer about the values that appear on the terminals, not the directionality of the device. The adder imposes a constraint on the values that can appear at its terminals. One way to implement a constraint is with rules, as in [Sussman80]:

A. IF input-1 is X and input-2 is Y, THEN the output is X+Y.
B. IF input-1 is X and the output is Z, THEN input-2 is Z-X.
C. IF input-2 is Y and the output is Z, THEN input-1 is X.

But if the adder is not known to be behaving correctly, *any* combination of values might appear at its ports, i.e., the constraint is suspended.

A suspect is consistent only if it is consistent for all other components to be behaving correctly. In constraint suspension, a suspect is checked for consistency by suspending its constraint and enabling the constraints associated with all other components in the device. When any contradiction arises, the suspect is ruled out: it cannot explain all the observations. For consistent suspects, constraint suspension also makes hypotheses more specific by computing how the suspect must have misbehaved. If no such misbehavior can be found, the suspect is inconsistent. Hence the technique can rule out many potential misbehaviors of a suspect at once.

Consider an example from [Davis84]. The predicted outputs of this device were F=12 and G=12, but instead F=10 was observed. By tracing dependencies the suspects are found to be ADD-1, MULT-1, and MULT-2.



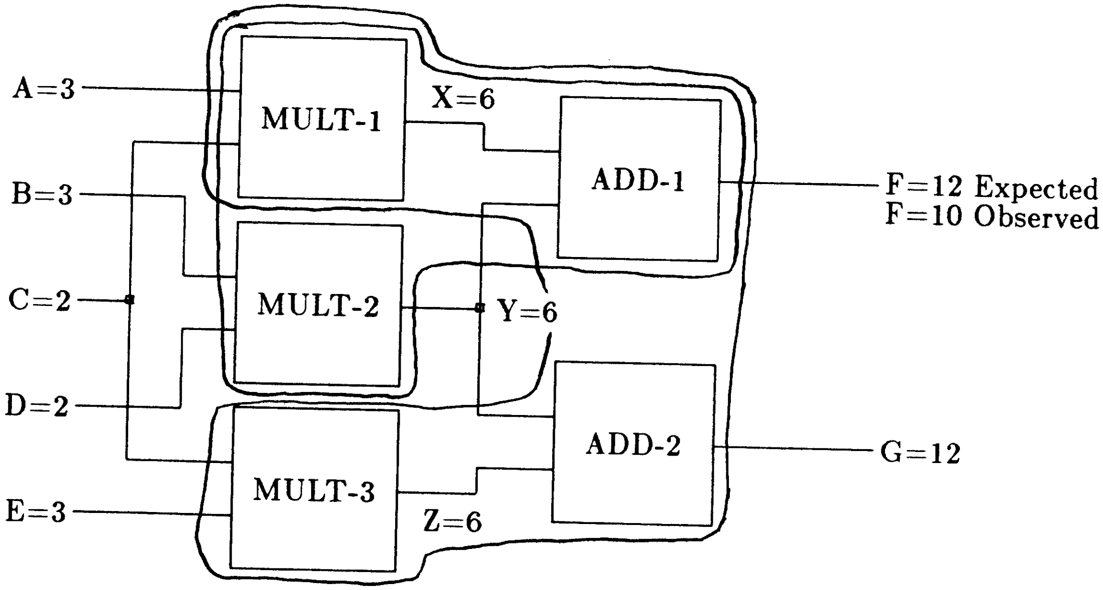
Second Adder-Multiplier Example

Because MULT-3 is not a suspect, Z=6; then, because ADD-2 is not a suspect, Y=6. Each suspect can now be checked for consistency by assuming that the other components are OK. To check whether ADD-1 is faulty, we reason as follows: since MULT-1 is OK, X=6, since MULT-2 is OK, Y=6, hence the adder is misbehaving by adding 6 and 6 to get 10. If MULT-1 is faulty, then ADD-1 and MULT-2 are not, hence X=4 and the multiplier is misbehaving by multiplying 3 by 2 to get 4. Finally, if

MULT-2 is faulty, MULT-1 and ADD-1 are not, hence X=6 and Y=4: but the latter is inconsistent with the earlier deduction that Y=6, therefore the suspect MULT-2 cannot explain the observations and is exonerated. The procedure not only rules out all failures of MULT-2 at one stroke, but also produces useful information about exactly how ADD-1 and MULT-1 are misbehaving if either of them is the true culprit.

LOX uses a similar procedure, but also interleaves the generation and checking of suspects, occasionally allowing it to exonerate all the predecessors of an exonerated suspect. The underlying intuition is that if the suspect can't explain all the observed discrepancies, then neither could its predecessors. This intuition is correct only in the absence of reconvergent fan-out. The system diagnosed by LOX has enough components (about 2000) and its structure is sufficiently free of reconvergent fan-out that the check for this special case turns out to be advantageous. A similar optimization is done by LOCALIZE with its 10,000 components organized into largely fan-in-free structures.

INTER and the ATMS troubleshooter perform a computation similar in some ways to constraint suspension. As in constraint suspension, all observations propagate their consequences uniformly throughout the device. Each discrepancy can result in conflicts with the consequences of other observations. Hence there can arise several overlapping sets of conflicting assumptions, i.e. several sets of components, each of which must contain at least one faulty component. Each such **conflict set** may be rediscovered several times, in contrast to constraint suspension, which in effect stops after finding the first conflict. The figure below shows the conflicts that this procedure discovers in the adder-multiplier example. The intersection of these sets of conflicting assumptions is the set of consistent suspects, MULT-1 and ADD-1.



Adder-Multiplier Example Conflict Sets

Saving conflict sets allows straightforward generalization to finding consistent hypotheses about independent multiple faults by using set cover instead of intersection, as is done through a variety of mechanisms in [First82,Reggia83,Reiter85,deKleer87]. Any collection of components that contains at least one element from each conflict – i.e. any set cover – would explain all the discrepancies. By Occam's razor, the preferred hypotheses are the *minimal* set covers, i.e. those set covers having no subsets that cover all discrepancies. Note that different set covers can be minimal and yet have different sizes; the notion of minimality has to do with preventing the inclusion of extraneous suspects, not with cardinality. Note also that constraint suspension as described above could be generalized to hypothesize and check hypotheses about multiple faults by suspending n-tuples of constraints, but without an explicit requirement that all conflict sets be covered, such a generator would be needlessly unconstrained.

The purpose of hypothesis checking is to exonerate suspects. We have seen five techniques for performing this check. The ordering below reflects increasing generality due to differing information requirements:

1. Directly exonerating components by reasoning from corroborations. This requires that the components in the domain have exceedingly simple behavior. LOCALIZE, INTER, and SOPHIE used this technique in certain cases.

2. Fault envisionment (used in SOPHIE and IDS) requires the use of built-in fault models for each component, and compares the simulation results to all observations.

3. Covering of suspect sets derived only from discrepancies requires the same information as hypothesis generation. It is used by DART and [Ginsberg84]'s related framework for multiple faults.

4. Constraint suspension, implemented in different ways in Davis' program and LOX, relies on the ability to infer components' inputs from their outputs.

5. Covering of suspect sets derived from conflicts involving both discrepancies and corroborations (as implemented in the ATMS troubleshooter) has the same information requirements as those of constraint suspension, but has important advantages for diagnosing multiple faults.

2.3 Hypothesis Discrimination

It is unlikely that an initial set of observations will be sufficient to yield a unique fault hypothesis. These competing hypotheses can be discriminated by probing – examining previously unobserved terminals – or testing – changing the device's inputs and reexamining its outputs. We first consider probe selection, then explore test generation.

2.3.1 Probe Selection

Assuming uniform cost for all probes, probes should be ordered in a way that minimizes the expected number required. Probe selection is based on the insight that each competing fault hypothesis may imply a different set of predictions about the device's response to its current inputs. Picking the best probe thus means selecting that one whose set of outcomes and resulting candidates are expected to minimize the number of probes that will come after it.

Probe interpretation – updating outstanding hypotheses given the results of a probe – is an extension of hypothesis checking. When using fault envisionment, the result of a probe is checked for consistency against the predicted value of that quantity under each possible fault hypothesis. When using constraint suspension, the result provides information that allows more constraints to apply, narrowing the number of consistent candidates. For efficiency, each candidate should cache sufficient information to reconstruct the consequences computed for it, i.e., the consequences of believing that all components not appearing in the candidate are normal. This suggests that an ATMS is an advantageous organization, since each predicted value can be indexed with the collection of working components that support it.

Since probes are generally considered expensive, it is usually a good idea to spend extra computation to select the most informative place to probe. We distinguish here two basic approaches: structural approaches, which use mainly information about device structure to select a probe point, and more elaborate decision-theoretic approaches, which ignore device structure and instead use estimates of likelihood for the various outstanding hypotheses and probe outcomes.

Structural approaches to probe selection are motivated by the insight that competing suspects tend to lie in groups. As a result, if information about values on the connections between suspects were known, the hypotheses could be discriminated quickly. Consider the now familiar adder-multiplier example above. MULT-1 and ADD-1 are suspects, and they could be discriminated if the intervening value at X were known. Nothing needs to be known about the behavior of the components; the fact that X is a good place to probe is plain from the structure.

A simple structural approach is to probe upstream from discrepant outputs until a correct value is found; at that point the component with the just-observed correct input will have an incorrect output, and hence is the faulty component. Components with multiple unknown inputs are searched depth-first; the search is terminated along any branch where the expected value is encountered. This is the essence of the *guided probe* algorithm [Breuer76].

Additional power is available from the observation that probe results introduce new discrepancies or corroborations at internal connections. Hence, each new probe result can eliminate more candidates than the conservative guided-probe strategy allows. Strategies to take advantage of this have been widely investigated.

INTER [deKleer76] achieved quick reductions in the number of candidates with a heuristic resembling binary search. INTER stored with each predicted value the set of assumptions that supported that value. In practice, each set of assumptions amounted to a list of suspects. Expected values that were still unobserved were then examined to see how many suspects contributed to their expected values. A probe point was then selected whose list of contributing suspects is nearest in length to half the number of outstanding suspects. The heuristic works well because the single measurement has a good chance of further constraining multiple hypotheses, indeed ruling some out completely.

Structural approaches to probe selection enjoy the advantage of simplicity: they use only information already available for hypothesis checking and result in reasonable strategies with a minimum of computational overhead.

A different approach to probe selection arises from the viewpoint that any given set of hypotheses represents some quantifiable amount of information about the broken device. Larger sets of hypotheses have more ambiguity and hence contain less information. The objective of probe selection is to select the probe that maximizes the expected information (hereafter, I). [deKleer87]'s program uses this approach, which involves six elements:

1. An information evaluation function assigns a score to an the current set of competing candidates. Roughly speaking, a good measure of the information contained in a set of candidates should be inversely proportional to the number of probes needed to narrow it to a single hypothesis.

2. A cost criterion can be used to weight the expected I and bias the selection of probes toward those that are easier.

3. All the possible outcomes for each probe must be enumerated. The outcome of a probe is the value observed in the device, and the possible outcomes are the ones predicted by candidates. In the adder-multiplier example, the outcomes of a probe at X are 4 (if MULT-1 is broken) and 6 (if ADD-1 is broken).

4. The consequences of each outcome from a given probe must be predicted. Some outcomes will reduce the size of the hypothesis set. After making a probe at X, for example, either MULT-1 or ADD-1 will have been exonerated. The value of I that would be obtained after each outcome measures the desirability of that outcome.

5. The probability of each candidate is needed. A simplistic approach under the single-fault assumption is to let each outstanding hypothesis be equiprobable. Another technique is to assign fault hypotheses their *a priori* probabilities. In certain domains domains this can be quite powerful. In troubleshooting digital hardware for example, RAM chips fail far more frequently than chips that contain only a few gates, due largely to the RAMs' greater complexity. Knowing this can

be helpful even if the actual failure rate estimates used are inexact.[2]

6. A probability must be assigned to each outcome. A simple approach is to assign to each outcome the sum of the probabilities of the fault hypotheses with which it is consistent. An important detail is that not every fault hypothesis implies a value for each measurable signal, hence a single hypothesis may be consistent with many outcomes. [deKleer87] deals with this by computing both lower and upper bounds on outcomes' probabilities.

Using these six elements, it is straightforward to compute for each possible probe a weighted expected value of I. The best probe to perform is the one that maximizes the expected value of I, i.e., that on the average will achieve the greatest reduction in the size of the hypothesis set relative to its cost.

The advantage of the decision-theoretic approach is one of generality: it provides a way to use information about both *a priori* failure rates and the relative difficulty of different probes, and generalizes to multiple independent failure failures. Given accurate prior probabilities it yields an optimal probing strategy on average.

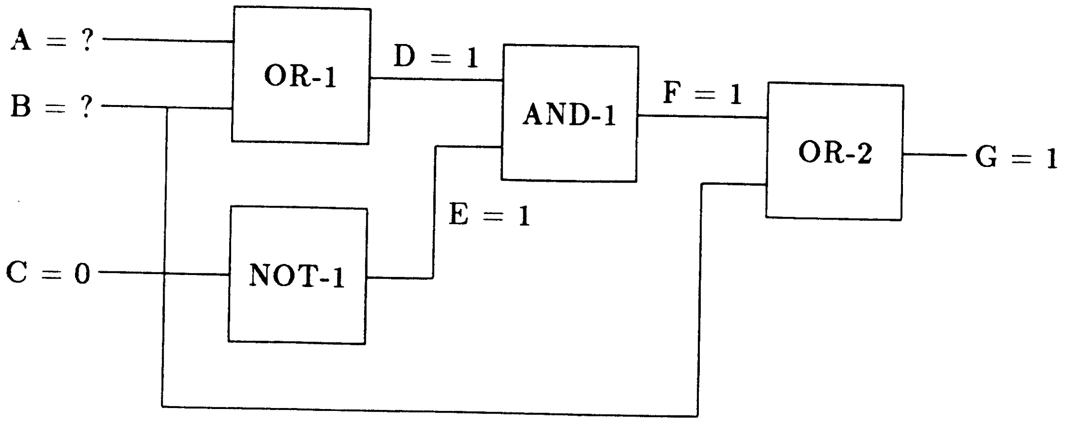
2.3.2 Test Generation

Test generation attempts to find a set of inputs that will yield new symptoms with different suspect sets. When this succeeds, we can gain new information that will help constrain the location of the faulty component. We focus our attention on the problem of test generation in the domain of digital circuits in the belief that the principal observations generalize.

The goal of traditional test generation techniques is to produce tests with *coverage* of many faults rather than diagnostic *specificity*. A test can be said to have specificity if it indicates an error only when one of a very small number of components is faulty. Achieving specificity in the presence of a number of competing fault hypotheses is akin to the problem of choosing a good probe point: the idea is to gather information whose results, positive or negative, will eliminate as many hypotheses as possible.

Shown below is a simple test generation problem to illustrate the ideas.

[2] One pitfall is worth mentioning: the probability of failure of a component may depend on the functional role it is playing in a device. In the electronics domain, for example, resistors tend to short out more frequently when used to sink large currents, or when sudden voltage surges are a threat. Failing to take account of such information in assigning failure probabilities to components may negate some of the advantage gained.



Simple Test Generation Example

Suppose one fault hypothesis is that AND-1 is misbehaving by responding with a 0 when both its inputs are 1. To construct a test focusing on AND-1, it is necessary to achieve 1's on both its inputs, hence achieve a 0 at input C, and achieve a 1 on either A or on B. Suppose we choose B=1. Now the 1 must be propagated from F to G. Ensuring that the output of OR-2 is sensitive to F requires a 0 on B, requiring backtracking to the previous choice of B=1, and assigning 1 to A instead. The resulting test assigns A=1, B=0 and C=0, expecting G=1 if AND-1 is unfaulted.

This combination of local propagations and backtracking is the essence of traditional test generation methods [Breuer76]. Propagating the expected outputs of the tested component to observable outputs is termed *path sensitization*; this involves the achievement of enabling values along the way. Achieving of values at the inputs of the tested component is termed *line justification* and can be viewed as propagation of values upstream, with choices to be made and backtracking required when conflicts arise. Such algorithms are exponential in the number of components (indeed, test generation for boolean circuits is NP-complete). Stated another way, test generation is a conjunctive planning problem in which the different goals mutually constrain one another.

Heuristic methods and dependency-directed backtracking have been applied to test generation by a number of researchers, e.g. [Rutman72], [Breuer79], and [Genesereth84]. [3]

Not every test has diagnostic value. Ideally, the expected output will rely on some, but not all, of the outstanding suspects. Just as in the probe selection problem, ideally the value examined should depend on about half of the suspects,

[3] DART's propositional representation of devices and the use of resolution residue obscures the algorithm somewhat.

and depend on at least one of them to behave in the same way it was supposed to in the original symptom case. Due to the difficulty of test generation, however, a test with *any* diagnostic value is usually acceptable. DART, for example, keeps trying to generate tests until it finds one that might possibly reduce the number of suspects, and uses that.

A more direct approach to ensuring diagnostic value is to select exactly one suspect as the focus of the test, and guide the procedure so that the test being generated involves the fewest other suspects as possible, and ideally no others. Note that it is impossible to always generate a test that relies on only one suspect. Indeed, it may be suboptimal anyway in light of observations made earlier about good probing strategies. This approach toward generating tests is illustrated in [Shirley83] which uses a number of heuristics for avoiding or neutralizing the effects of suspects other than the focus.

Using these heuristics, [Shirley83]'s program is usually able to produce tests that rely on only a subset of the suspects, and hence have diagnostic value.

2.3.3 Summary

The purpose of both probe selection and test generation is to add new information that allows consistency checking to exonerate additional candidates. Depending on what is possible and cost-effective in the domain, either probing or testing may be used to gain this additional information, the common theme being that the best action can be selected on the basis of how it is expected to affect the remaining hypotheses. Different techniques make use of different information and yield different results:

1. The guided probe technique can be used when possible failures are treated as equally probable, and the cost of additional probes is proportional to their distance from previous probes.

2. Probe selection based on comparing sets of assumptions underlying various predictions can be used when failures are equally probable and probes have equal cost.

3. Probe selection based on decision analysis subsumes a variety of strategies. It can make use of all available quantitative information about relative failure rates and probe costs, and can be generalized to deal correctly with multiple faults.

4. Test generation via search requires information about ways to achieve desired values on individual component outputs. The combinatorics of the problem also requires that heuristic guidance be provided to focus search toward those primary inputs most easily achieved.

5. Test generation can also benefit from heuristics that try to prevent the test of a particular suspect from depending on the proper functioning of competing suspects. Such tests are more likely to yield different suspect sets and hence have discriminatory power.

3 Assumptions and Limitations

The effectiveness of model-based diagnosis is inextricably bound to the appropriateness of the models it is provided. Models of structure and behavior, like all representations, involve simplifying assumptions; in this case the assumptions affect both the completeness of the hypothesis generator and the discriminatory power of the hypothesis checker. In the following section we discuss these assumptions, focusing on those that are fundamental in the sense that to abandon them would result in uninformative or impractically expensive computations. We also present some guidelines about useful assumptions to make – in effect, some general principles about constructing good models for troubleshooting.

3.1 The Completeness of the Hypothesis Generator

As noted earlier, a complete set of fault hypotheses can be generated trivially by enumerating all components. But this or any other set of components is only complete with respect to the model, not with respect to the real device. There are two ways a hypothesis generator might be incomplete in this broader sense: (i) a possible fault location is not represented among the components: or (ii) some real interaction between components is not represented among the connections. Both mistakes arise inevitably from built-in assumptions, often made because they are realistic, but no less limiting.

3.1.1 Components Represent the Possible Fault Locations

Fault hypotheses generated by the methodology described above take the the form of specifying one or more components that might be misbehaving. Hence, to choose which parts of a device get represented as components is to choose which fault hypotheses can be generated. Consider for example a circuit board, which can fail because a piece of metal etch is cracked. If the program is to diagnose that fault correctly, then the metal etch itself should be represented as a component, otherwise the program will fail to generate the hypothesis.

The process of elaborating the model to include more and more fault locations need not be endless. Pragmatic limits on the level of detail that needs to be included arise from the environment in which the automated troubleshooter operates. The following two principles apply in general:

- The level of detail that a model includes should be limited by the possible repairs. For example, there is little point in distinguishing the individual transistors on a chip as separate components, since chips aren't usually repaired.

- The level of detail should be limited by the distinguishability of the effects of the faults. For example, if two wires run in parallel for some distance, and all that the troubleshooter can do is measure voltages at one end, then shorts between the wires at all points along that distance are indistinguishable in their effects and can be represented as a single possible short.

Even given a representation that is complete in this respect, however, the representation of device structure as a graph, with the components represented as nodes and connections between components represented as arcs, still reflects a bias about the kinds of faults that will be represented. The representation doesn't lend itself to representing faults that arise from the presence of things that shouldn't be present. For example, boards can fail because a spurious solder splash introduces a connection between functionally separate signals (a "bridge fault"). Naively extending the representation of structure to diagnose such faults would result in adding pseudo-components to represent the absence of solder – or, conversely, the presence of gaps – between every pair of wires. While possible in principle, the idea is counterintuitive and combinatorially explosive.

Fortunately, it isn't necessary to represent all such fault locations explicitly; it is only necessary that the hypothesis generator propose them. The fault locations can be represented implicitly in the graph, and created as needed by the hypothesis generator from another representation. The intended presence of gaps between wires, for example, can be derived from a representation of the physical layout of the board, as in [Davis84].

3.1.2 Connections Represent Interactions

Similar remarks apply to the connections that appear in the representation of device structure and behavior. Just as the notion of "component" can be generalized to the notion of "potential fault location," connections can be used to represent any kind of interaction. Because hypothesis generation marks as suspects only those components reachable by following connections, any missing interaction between components means a possible loss of generator completeness, too.

For example, representing the behavior of components as having a single direction of cause and effect is a useful abstraction for design purposes. Most digital devices can be viewed this way and this abstraction is useful in diagnosis because it reduces the number of suspects generated from each discrepancy. But it can be violated when components fail. Components can in fact influence their inputs, e.g. a faulty gate can ground its inputs. Diagnosing such faults correctly requires a model of the device that takes into account the fact that gates interact not only through voltage, but also through current. More striking, in any device there are many electromagnetic and thermal couplings between components that can profoundly influence their behavior, and yet are virtually never represented explicitly. For example, high frequency signals on adjacent wires can interfere with each other, but electrical schematics don't normally show this interaction. nor the shielding that is used to reduce it.

Ideally, the pragmatics of the tools available to the troubleshooting program could be used to dictate the limits to the level of elaboration needed, as discussed earlier. However, this appears to be more difficult to do for interactions than for components. For example, it would appear that interactions that can't measurably influence behavior can be ignored. But "measurable influence" can be cumulative; for example, while it is safe to assume that any given pair of gates on a chip don't interact through their power connections, all the gates on the chip together may draw enough current to cause fluctuations in the power supply voltage. Such phenomena are notoriously difficult to anticipate in engineering models. Since the problem is one of modeling, model-based diagnosis inherits it.

3.1.3 Controlling Hypothesis Generation

A model that included all the connections through which components might possibly interact would leave hypothesis generation underconstrained. Assume for a moment that we were willing to temporarily sacrifice some completeness in the generator, in return for the ability to generate fault hypotheses in a more constrained way. Those models that provide the most constraint on hypothesis generation can be characterized as follows:

- Models with sparse and unidirectional connections constrain hypothesis generation. When there is an identifiable direction of information flow in the device, a model that assumes that the direction of flow is preserved in the malfunctioning device will generate fewer suspects than a model in which the information flow is not assumed to be preserved.

This principle appears implicitly in most of the programs surveyed. LOX and LOCALIZE in particular diagnose systems with hundreds or thousands of components successfully largely because the systems involved can be modeled as having relatively sparse and mainly unidirectional connections. These programs build in the assumption that whatever the underlying malfunction is, the intended directionality will be preserved.

Another way of controlling hypothesis generation is to use a hierarchic device model, as in [Davis84] and [Genesereth84]. The program can generate and check suspects among components at higher levels before examining their subcomponents. Hierarchy is especially useful when it is strict and a single failure is assumed, since all the subcomponents of an exonerated component are exonerated as well:

- A model should be hierarchically organized, with strict decomposition of components where possible.

A generalization of this idea is to start with a description of structure and behavior adequate only to represent the most important faults. Faults that occur "outside" that model will typically result in what appears to be intermittent or multiple faults. For example, a digital gate that pulls down all its input signals can appear to be caused by multiple faults in the gates that are supposed to drive those signals; a bridge between wire X and wire Y can make both X and Y appear intermittently grounded. When the only consistent explanation of a particular set of symptoms seems to be multiple independent "normal" faults, an alternative, simpler explanation can be sought in a second model adequate for representing more unusual faults. Second and succeeding models can represent different fault categories among their components and connections. This is done in Davis' program with two models: the initial hierarchic model represents only wires, boolean gates, and compositions thereof; a second model includes physical layout information, from which possible bridge faults can be hypothesized.

This approach leaves some issues unresolved. With a variety of different models appropriate to different fault categories, it is unclear in what order the program should try the models. One possibility is to try those that include the most *a priori* probable fault categories. Another would be to try those that are simpler, perhaps as measured by a count of components and connections. Ideally, the program should choose an appropriate model based on the particular symptoms at hand, though the relevant criteria for such a choice is unclear. Nevertheless, a useful principle remains:

- Layered models can be used to ensure that the simplest hypotheses are explored first, while retaining completeness overall, as each successive layer includes additional faults.

3.2 The Discriminatory Power of the Hypothesis Checker

The job of the hypothesis checker is to determine whether fault hypotheses are consistent with all the observations of the device. The discriminatory power of the checker is determined by its effectiveness in distinguishing between consistent and inconsistent hypotheses. There are three reasons why current diagnosis programs fail to detect inconsistencies and thereby fail to yield unique diagnoses: (i) the computational machinery is weak because it is usually based on local, component-centered propagations (ii) some constraints present in the world are not represented effectively in the device model (iii) the device is modeled in such a way that the problem is inherently underconstrained.

3.2.1 Detecting Global Inconsistencies via Local Propagation

In its most general form, checking the consistency of a fault hypothesis is a constraint satisfaction problem – we wish to find out whether or not there exists an assignment of values to all the terminals in a device such that they are consistent with the observations and with each other. For efficiency reasons, most of the programs surveyed here rely on local propagation to solve this problem and hence make inferences about one value at a time. A characteristic of all such approaches is that they cannot always compute all the consequences of the observations; as a result, contradictions may go undetected, resulting in the inappropriate survival of inconsistent hypotheses.

This incompleteness typically occurs when a collection of constraints, each involving n values needs $n-1$ of those values assigned before it can deduce the last. Such simultaneities occur in rings of constraints when each constraint has only $n-2$ of its values assigned. One possible effect of the simultaneity is that even though there is only one consistent set of assignments for the group, this goes undetected. Simultaneities are common in non-directional domains and arise in directional domains in structures with reconvergent fanout.

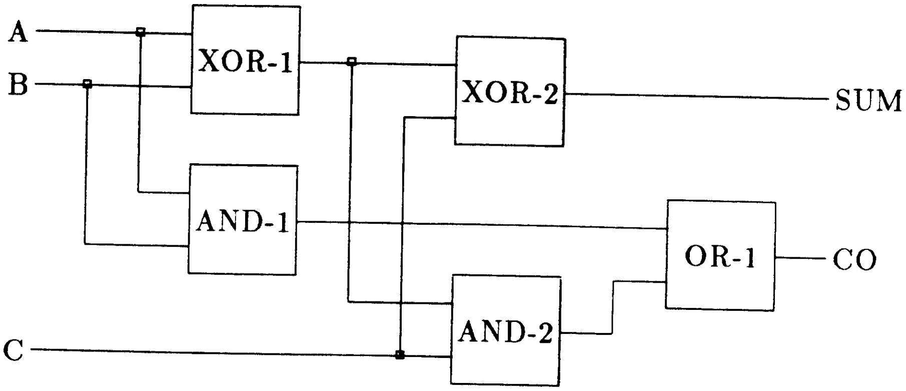
Simultaneities are amenable to a variety of techniques, including (i) relaxation, as in the Gauss-Seidel method for solving linear systems, (ii) enumeration over finite sets of possible assignments, (iii) propagation of symbolic expressions [deKleer80], or (iv) addition of additional constraints, perhaps encapsulating several components ("slices" [Sussman80]). Relaxation techniques are appropriate in continuous domains. The second technique can be viewed as adding the capabilities and attendant control problems of a full first-order theorem prover with equality. Similarly, the third may involve an algebraic manipulator of considerable complexity (e.g.MACSYMA). The technique of adding explicit nonlocal constraints, in contrast, requires no additional propagation machinery, although it complements (i)-(iii). Encapsulating groups of components with nonlocal constraints places the burden of deadlock avoidance on the device model instead. This suggests another guideline for a good model:

- Organizations of components that are likely to cause local propagation simultaneities, e.g. structures with reconvergent fanout, should be encapsulated to break impasses wherever possible.

3.2.2 Hierarchy, Abstraction, and Constraint

The most straightforward way to use nonlocal constraints is to organize components into a hierarchy, so that each component in the hierarchy has its own constraints. These constraints may make use of behavioral abstractions not available at lower levels of structural detail. One common source of such a hierarchic description with its accompanying behavioral abstractions is the device's design description.

The gates shown below, for example, are designed to function as a full-adder. The full-adder's composite behavior description is almost as simple as those of its individual gates: the output, viewed as a 2-bit integer. is the sum of the inputs, viewed as 1-bit integers. The vocabulary of integers, as opposed to bits, simplifies reasoning about the constraints on this group of gates. For example, the full-adder constraint can include a rule such as "if both outputs are 1, then all three inputs are 1." This relationship would be difficult for a purely local constraint propagator to discover from the gate level description. Other techniques for discovering such relationships, such as constructing the truth table of the device, are combinatorially impractical. The essential step is in choosing a vocabulary in which the behavior becomes simple to express, but that choice appears difficult to automate.



Full-Adder Structure

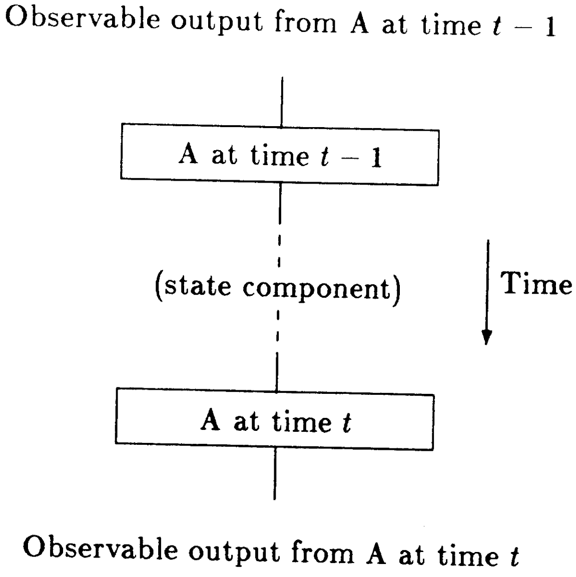
This example illustrates a particularly important way that a design hierarchy can add useful constraints: abstraction can make it easier to infer component inputs from their outputs. This helps all approaches to hypothesis checking (constraint-based or otherwise) to detect inconsistencies. While "inversion" of behavior is straightforward for simple components, components with many terminals or with internal state are more challenging. If as a consequence of behavioral complexity the knowledge is incomplete, i.e., constraints that invert behavior are missing, not all contradictions will be discovered. Another characteristic of a good model, then, is:

- Hierarchic decomposition should facilitate making inferences about components' inputs from their outputs.

3.2.3 Hidden State

Devices whose components have time-dependent behavior can in principle be modeled and diagnosed no differently from static devices. If behavior is described by rules, for example, the rule language can be extended to include delayed responses and other kinds of dependence on prior states. Hypothesis generation and checking for such devices follows the familiar outlines, but a fundamental difficulty arises when components have "hidden" state. In a memory chip with 1024 1-bit words, 1023 are hidden in the sense that the state can only be examined one word at a time. The presence of hidden state typically results in inherently underconstrained problems: competing hypotheses cannot be discriminated because of ambiguity about the device's internal state.

[Hamscher84] presents one example of this phenomenon in the digital domain. To check whether a particular component could have misbehaved in a way that not only explains all the observed discrepancies, but that is also nonintermittent, requires inferring what its inputs and outputs must have been at every time step. If the inputs to a suspect depend upon its behavior at a previous time, and it is not possible to observe its intermediate state, it is impossible to rule out the suspect; the problem is inherently underconstrained. The figure below illustrates this abstractly.


Observable output from A at time $t-1$
A at time $t-1$
(state component)
Time
A at time t
Observable output from A at time t

Unobserved State

If A is a suspect, but we know only its inputs at time $t-1$ and its output at time t, checking whether A is a consistent suspect requires inferring its output at $t-1$ and inputs at t. To do this, however, requires knowing A's behavior, which is unknown because it is a suspect. The problem is analogous to solving a system of n linear equations in $n+1$ unknowns; it is inherently underconstrained. As noted, the only way to solve this is to add additional observations, preferably of the intermediate state between $t-1$ and t. Similar remarks apply to domains in which components' states change continuously rather than discretely.[4]

The inherent ambiguity of collections of components with hidden state suggests that for pragmatic reasons, levels of detail at which the distinct components are visible should be suppressed. For example, a group of components that can't be discriminated among using the observation tools available to the diagnosis program should be abstracted into a single component with simple behavior. In principle, it is possible to describe any device at such a behaviorally and temporally abstract level that delay can be ignored, feedback loops encapsulated into primitive components, and hidden state abstracted away. While a completely state-free model may discard too much detail, the following guideline still offers useful assistance:

- A good model minimizes hidden state.

[4] Having components with hidden state also increases the computational cost of generating discriminating tests. Achieving a particular set of inputs at an embedded component, for example, might require finding a complex input sequence that sets the states of certain components without disturbing others.

4 Conclusion

Existing programs for automated diagnosis of devices from first principles have much in common despite apparent differences of domain and mechanism. They have similar procedures to generate and check fault hypotheses, and similar limitations due to their representations of the devices they must diagnose. The first similarity suggests that domain-independent diagnosis from first principles is within reach. The second indicates that there remains a substantial agenda of open problems.

Fault hypotheses are typically generated by examining a trace of the expected behavior of the device. Hypotheses are then checked for consistency either by explicit simulation or by attempting to deduce a specific component misbehavior by reasoning from external observations back to the embedded component.

The effectiveness of both phases depends crucially on the device models. For example, the completeness of the generator depends on the level of detail of the components; the number of hypotheses generated for each discrepancy depends on the type and density of component connectivity; the power of the reasoning machinery that rules out inconsistent hypotheses depends in part on whether inferences about components' inputs can be made from their inputs.

Substantial problems remain to be addressed. Most of the programs work on "toy" examples, and there is evidence to suggest that scaling up to deal with complex and highly connected devices may be difficult, both from the standpoint of computational complexity and from the standpoint of knowledge engineering. General principles for constructing good models exist, but they remain few, sketchy, and in some cases contradictory because of the difficulty of reconciling the underlying goals of ensuring completeness while utilizing the constraints that the troubleshooting domain provides.

Acknowledgements

Discussions through various media with Meyer Billmers (DEC), Michael First (Columbia), Michael Genesereth (Stanford), Harold Haig (MIT), Tom Knight (Symbolics), Willie Lim (MIT), Jeff Pan (Schlumberger), Ramesh Patil (MIT), Charles Rich (MIT), Mark Shirley (MIT), Howard Shrobe (Symbolics), Reid Simmons (MIT), Ethan Scarl (MITRE), Peter Szolovits (MIT), Daniel Weld (MIT), Brian Williams (MIT), Raúl Valdés-Péres (CMU), Jeffrey Van Baalen (MIT), and Peng Wu (MIT) were helpful. Discussions with Johan deKleer (Xerox PARC) were especially helpful. Acknowledgement does not imply agreement with opinions stated herein.

References

[Breuer76] Breuer, M. and A. Friedman. *Diagnosis and Reliable Design of Digital Systems*, pp. 147-149. Computer Science Press, 1976.

[Breuer79] Breuer, M. A. New Concepts in Automated Testing of Digital Circuits. In *Computer Aided Design of Digital Electronic Circuits and Systems,* pp. 57-80. North-Holland Publishing Company, Brussels and Luxembourg, 1979.

[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.

[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.

[deKleer80] de Kleer, J., and G. J. Sussman. Propagation of Constraints Applied to Circuit Synthesis. *International Journal of Circuit Theory* 8(2):127-144, April, 1980.

[deKleer87] de Kleer, J., and B. C. Williams. Diagnosing Multiple Faults. To appear in *Artificial Intelligence*, 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.

[Ginsberg84] Ginsberg, M. L. *Counterfactuals.* Stanford Knowledge Systems Laboratory Report KSL-84-43, Department of Computer Science, Stanford University.

[Hamscher84] Hamscher, W. C., 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.

[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.

[Reggia83] Reggia, J. A., D. S. Nau, and P. Wang. A New Inference Method for Frame-Based Expert Systems. In *Proceedings of AAAI-83*, Washington, DC, pages 333-337.

[Reiter85] Reiter, R. *A Theory of Diagnosis from First Principles.* Department of Computer Science, University of Toronto, and the Canadian Institute for Advanced Research, December 1985.

[Rutman72] Rutman, R. A. Fault-Detection Test Generation For Sequential Logic by Heuristic Tree Search. IEEE Computer Research Paper R-72-187, 1972.

[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.