THE FIS ELECTRONICS TROUBLESHOOTING PROJECT

Frank Pipitone, Kenneth DeJong, William Spears, and Michael Marrone

U. S. Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory, Washington, DC

1. INTRODUCTION

FIS, short for Fault Isolation System, is an ongoing research project at the Navy Center for Applied Research in Artificial Intelligence, a branch of the U.S. Naval Research Laboratory. The focus of the work is a model-based expert system shell capable of acquiring from a user a description of a piece of electronic equipment, called a unit under test (UUT). This description is later used by FIS to perform such diagnostic functions as recommending the next "best" test to make on the UUT during a fault isolation sequence and estimating fault probabilities after each test is made. These and other capabilities make FIS a powerful tool that can be used in a variety of diagnostic settings, as discussed in Section 1.2. FIS has been developed primarily with large-scale analog-hybrid electronic systems such as radar and sonar systems in mind, but it is applicable at least in principle to any human-engineered system with discrete replaceable components. FIS is written in LISP and has been under development for several years. Earlier versions of the FIS prototype system are described by Pipitone [1,2].

1.1. The Need for Artificial-Intelligence-Based Diagnostic Systems

The maintenance of electronic equipment has drawn increasing attention during the past decade as a potential artificial intelligence (AI) application, particularly in the military. This has been motivated by the increasing complexity of military electronic systems and the resulting high maintenance costs as well as the scarcity of highly trained technicians.

The Navy has been particularly interested in and supportive of the development

THE FIS ELECTRONICS TROUBLESHOOTING PROJECT

Frank Pipitone, Kenneth DeJong, William Spears, and Michael Marrone

U. S. Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory, Washington, DC

1. INTRODUCTION

FIS, short for Fault Isolation System, is an ongoing research project at the Navy Center for Applied Research in Artificial Intelligence, a branch of the U.S. Naval Research Laboratory. The focus of the work is a model-based expert system shell capable of acquiring from a user a description of a piece of electronic equipment, called a unit under test (UUT). This description is later used by FIS to perform such diagnostic functions as recommending the next "best" test to make on the UUT during a fault isolation sequence and estimating fault probabilities after each test is made. These and other capabilities make FIS a powerful tool that can be used in a variety of diagnostic settings, as discussed in Section 1.2. FIS has been developed primarily with large-scale analog-hybrid electronic systems such as radar and sonar systems in mind, but it is applicable at least in principle to any human-engineered system with discrete replaceable components. FIS is written in LISP and has been under development for several years. Earlier versions of the FIS prototype system are described by Pipitone [1,2].

1.1. The Need for Artificial-Intelligence-Based Diagnostic Systems

The maintenance of electronic equipment has drawn increasing attention during the past decade as a potential artificial intelligence (AI) application, particularly in the military. This has been motivated by the increasing complexity of military electronic systems and the resulting high maintenance costs as well as the scarcity of highly trained technicians.

The Navy has been particularly interested in and supportive of the development

of fault isolation expert systems that can improve the quality of their maintenance and troubleshooting activities. As an example, Navy technicians on aircraft carriers may be responsible for troubleshooting several hundred different (sub)systems for which they have had varying amounts of training (frequently little or none). To compensate, the Navy has and continues to invest heavily in automatic test equipment (ATE) to aid or replace these technicians. The quality of the "test programs" that drive these ATE stations varies dramatically in spite of a uniformly high cost to acquire them.

Although it is tempting to leap to the conclusion that one could significantly improve this sort of troubleshooting activity with reasonably straightforward applications of current expert system technology, there are several aspects to the problem that raise significant technical issues. First, with several hundred different systems to maintain, it seems infeasible to think in terms of independently developed expert systems for each one. Rather one thinks in terms of a more general fault isolation shell providing a common knowledge acquisition-representation scheme for use with all subsystems. However, even with this level of generality, there are still several hundred knowledge bases to be built, debugged, and maintained in a context in which there can be considerable overlap and/or similarity in the content of many of the knowledge bases. These observations strongly suggest the development of a sophisticated knowledge acquisition system that can be used to facilitate the construction of a new knowledge base for a specific system in a variety of ways including reusing and/or adapting existing knowledge modules.

Compounding the problem of applying current expert system technology is the fact that, for many of the subsystems being maintained, there is little human expertise in the traditional sense of finding someone who is good at fixing a particular subsystem and capturing his or her knowledge in a set of associative rules. Rather, technicians depend heavily on the structural and functional descriptions contained in the technical manuals of the many subsystems they attempt to maintain. This suggests that simple rule-based architectures are not likely to be sufficient for the task at hand.

For the past few years, we have had the opportunity to explore the use of AI and expert system technology in this setting. We have built (and discarded) several prototype fault isolation shells in the process of understanding how this technology can be usefully applied. We feel that we have now evolved an architecture (FIS) that directly addresses the issues discussed above.

1.2. Maintenance and Troubleshooting Applications

Before diving into the details of FIS, it is important to understand how FIS might be used in maintenance and troubleshooting applications. The following paragraphs discuss what we feel are the most promising modes of use.

The first and most obvious use of FIS is as a technician's aid. The model we have in mind is a cooperative effort in which FIS and a technician *together* fault isolate complex systems. To be most effective in this mode, a tool must have a "mixed initiative" interface in the sense that it must be willing to accept and

integrate actions taken by a technician as well as being capable of recommending actions of its own when asked by a technician.

A second obvious use of FIS is as the controller of automatic test equipment (ATE). There are many continuing efforts to reduce maintenance and troubleshooting costs by removing humans as much as possible from the diagnostic loop. This is accomplished by developing ATE stations that are capable of making tests, interpreting test results, and making replacement recommendations. In the electronics world, such stations consist of a battery of signal generators and measurement devices that are controlled by a test program running on a small mini- or microcomputer. After an operator loads the appropriate test program and attaches the test gear to the UUT, the ATE system runs through a sequence of diagnostic tests in an attempt to fault isolate.

For the most part, these test programs fault isolate by following a decision tree designed by a test engineer at some point in the past. When these decision trees are accurate and up-to-date, they provide an efficient form of fault isolation. However, they are also notorious for their development costs, their rigidity, and the cost of keeping them up-to-date.

FIS can be applied to the ATE world in several ways. First, if a particular application requires the flexibility of *dynamic* decision making, FIS can be run directly on ATE gear with sufficient computing capability. However, if that is not possible or desirable, FIS can be used by test engineers to reduce significantly the cost of developing the required decision trees.

Even with the most sophisticated ATE stations, fault isolation can be hampered significantly by equipment that was designed without testability considerations in mind. Such problems arise for a variety of reasons including a lack of interest by design engineers and the effort involved in evaluating a design for testability. By applying FIS during the design cycle to generate fault isolation trees, many testability problems can be identified easily and inexpensively.

There are many contexts in which it is not possible or desirable to remove technicians from the fault isolation process. Rather, the emphasis is on providing fast, cost-effective training programs to improve technician skills. Here again we feel FIS can be used as an effective training tool by providing an environment in which simulated faults can be introduced and the fault isolation activities of trainees can be monitored and compared with those made by FIS.

1.3. Current Focus of FIS

Although the underlying architecture of FIS is general enough to support all the application areas discussed in Section 1.2, the Navy has been particularly interested in its use as a technician's aid and as a test tree generator for ATE stations. As a consequence, the first two interfaces actually implemented have been for these two application areas. Figure 1 illustrates how FIS is used in these contexts.

To be successful in either of these applications, the diagnostic components of FIS depend heavily on the quality of the underlying causal model of the UUT developed by a design/test engineer using the knowledge acquisition components

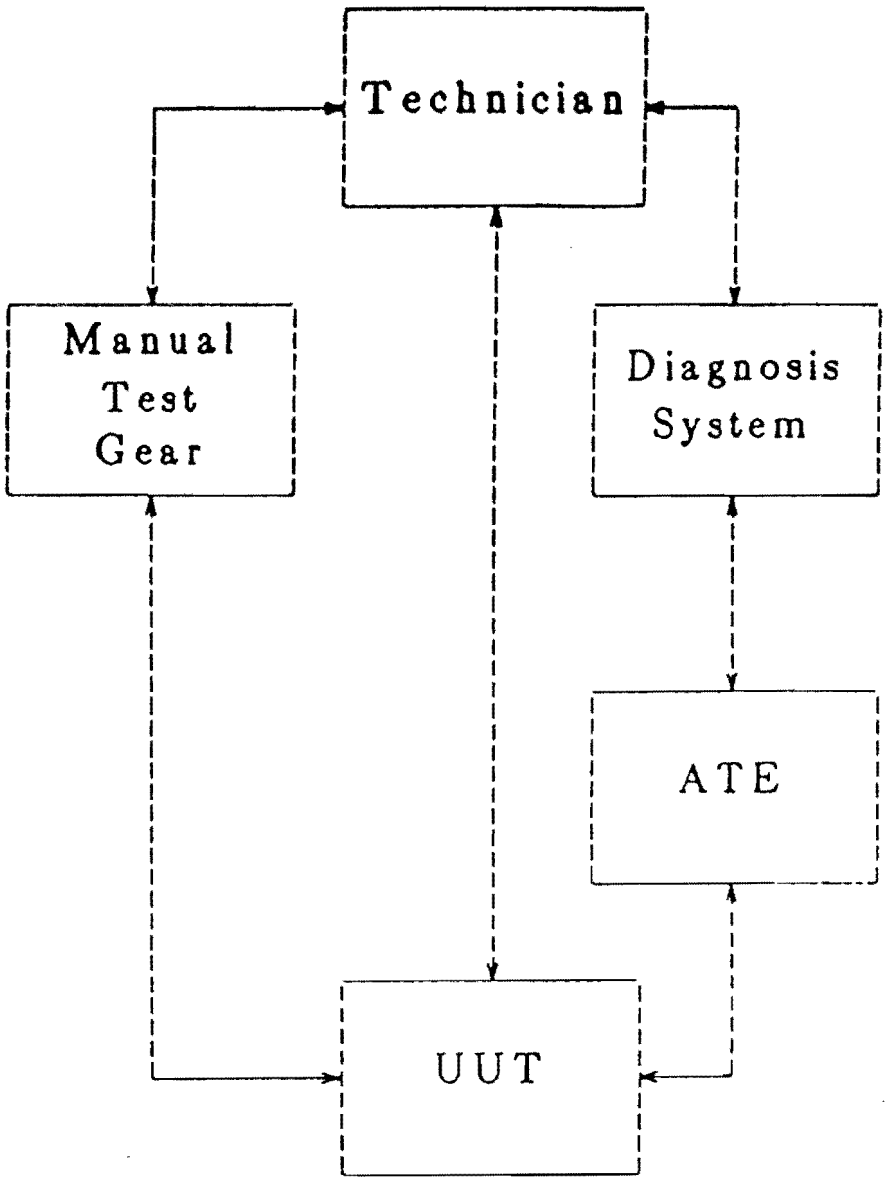


Figure 1. Application of a diagnosis system such as FIS.

of FIS. Ideally, one would like to start with the CAD/CAM database used by most manufacturing systems today. However, such databases are frequently proprietary or in highly nonstandard formats. As a consequence, we currently assume only that a design/test engineer has access to documentation describing the UUT. This includes schematic and functional block diagrams, specified values of measurable parameters at various test points, theory of operation, some notion of the relative failure rates of the replaceable components (modules), and the cost (in time) of various test operations with available test gear.

Given the above information, an engineer uses the knowledge acquisition interface of FIS to create a qualitative causal model of the UUT. FIS later uses this model in some diagnostic setting to recommend repeatedly a next best test for maximal informativeness per unit cost and to analyze the results of tests made until faulty replaceable modules are identified. These capabilities can be used to generate a test decision tree automatically. Also, FIS can answer queries about the UUT or

about the diagnostic process. Typical queries include what is the merit of a given test, what causal path led to the suspicion of a given module, what are the modules affecting a given test, what is the current probability of a given fault combination, and what are the causal rules describing a given module.

The initial focus of the FIS system can be summarized as follows:

(a) Provide the facilities for acquiring with minimum effort a UUT description suitable for diagnosis.
(b) Provide the reasoning mechanisms for recommending the next best test to make or the module to be replaced.
(c) Maintain good estimates of fault probabilities after tests.
(d) Respond to queries about the UUT model or explain (b) or (c).
(e) Automatically generate a test tree using (b) and (c).

1.4. Our Approach and Related Work

The basic approach to diagnosis used in FIS is that of capturing the important part of the behavior of a UUT for diagnostic purposes by describing each replaceable module with local causal rules and specifying the connectivity of the modules. The system then uses this network of causal rules to determine the set of possible causes (ambiguity set) of any failed test and which modules to certify in what manner when a test passes. This information is used in conjunction with a priori fault probabilities by an efficient Bayesian algorithm to estimate posterior probabilities of module faults and test outcomes. These are used along with test cost data and module replacement costs to make next best test or replacement recommendations. The next best test is found using a combination of fast heuristics to screen the available tests and information theoretic entropy (Shannon entropy) calculations to evaluate further the remaining candidates. No single-fault assumption is made. Also, we do not acquire global, empirical symptom–cause associations such as "frequency high at terminal7 implies module3 or module4 bad" during knowledge acquisition. FIS computes such relations from module descriptions and connectivity. The choices of method were strongly driven by the need to minimize knowledge acquisition cost. Effective automatic diagnosis can be achieved in many ways, but most approaches require much data about the UUT. Section 3.1 discusses some of the major alternatives.

The following is a summary of some of the more novel features of the FIS fault diagnosis system:

(a) The ability to do accurate diagnosis using a qualitative behavior model of a complex analog–digital UUT without simulation.
(b) An efficient UUT knowledge acquisition capability.
(c) An efficient evidential reasoning method specialized for device troubleshooting based on Bayesian principles.
(d) A natural treatment of multiple faults.

(e) An efficient method for computing the entropy of a complex system for use in best test selection.

Notable recent work has been directed at some of the same problems FIS has addressed. For example, DeKleer [3] gives a clear analysis of the problems of computing fault hypothesis probabilities, test result probabilities, and using entropy for test recommendation. However, this approach requires a strong UUT model; one must be able to predict module output values from module input values. It is often impractical to provide such a model for analog systems, although for digital systems it may be appropriate. Also, computational efficiency is not addressed. For the results to be applied, one needs fast probability and entropy algorithms and module simulations. Another recent work [4], based on Bayesian belief networks [5], provides a powerful mathematical approach to treating the joint statistics of component faults and signal abnormalities.

Much previous related research deals only with digital circuitry [6,7]. Most work on reasoning about analog devices is restricted to simple components because of the demands of performing detailed circuit analysis [8–10]. One that treats a complex analog system (a radio receiver) [11] requires considerable knowledge acquisition effort. Another [12] uses only a shallow functional model. Systems that have general analog-digital applicability [13,14] use only topological (connectivity plus directionality) knowledge about the UUT and therefore are forced to suspect all upstream modules from a failed test. Also, such systems cannot handle passed tests well because they cannot know to what extent each module upstream of a given passed test can be certified as good, lacking functional modeling. King [15] gives a survey of some previous work.

2. ARCHITECTURE OF THE FIS SYSTEM

The major components of FIS, as well as its two principal users, are illustrated in Figure 2. We will describe the components chronologically as they are used. First, the knowledge engineer, whose principal expertise is a detailed understanding of the type of equipment to be diagnosed, describes the UUT to the computer. He or she does this at a computer terminal via the knowledge acquisition interface (KAI), producing the UUT description, which is then stored in a file. A principal concern in the design of the KAI is to maximize the ease with which a user with a good electronics background but little familiarity with computer science can generate a description of the UUT that will yield acceptable diagnostic performance in FIS. This description contains such information as a priori rates of failure of the replaceable modules, costs (primarily in time) of setup and test operations, connectivity and qualitative functional descriptions, a set of allowed tests, and a graphics description for displaying a block diagram of the UUT.

The electronics library is primarily intended to store partial or complete qualitative functional descriptions of modules that occur frequently in the UUTs being described. The electronics library is to be referred to whenever possible while using

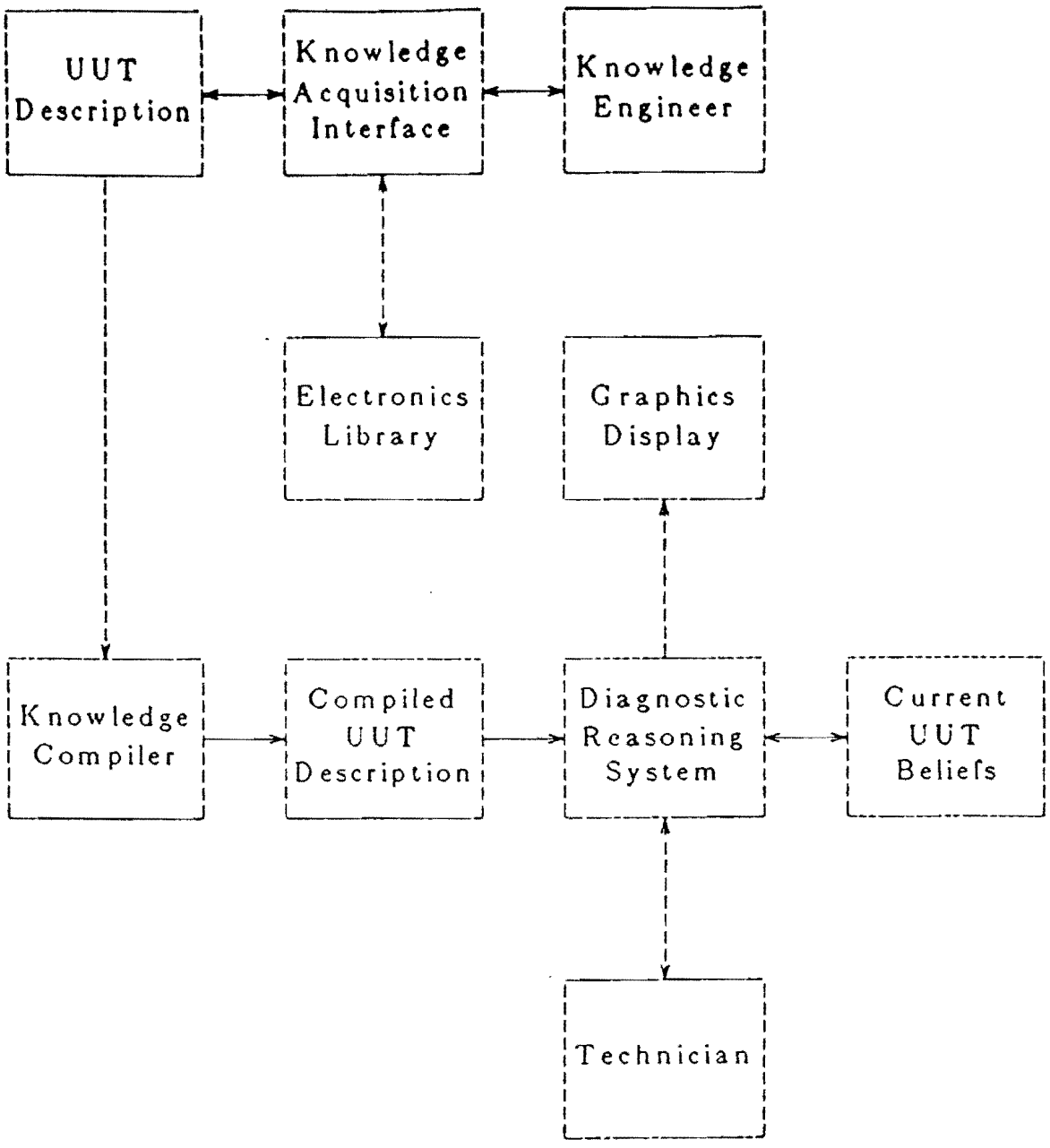


Figure 2. The architecture of FIS.

the KAI to enhance speed. The resulting approximate module descriptions can then be edited using the KAI. Also, the KAI can be used to add, delete, or modify items in the library. After its completion, the UUT description is processed by the knowledge compiler. This produces a file containing the compiled UUT knowledge.

The purpose of the knowledge compiler is simply to transform knowledge from a form suitable for editing to a form suitable for efficient diagnostic computation. The knowledge compiler is run after the UUT description is completely finished and stores the compiled UUT description in a file.

Once we have compiled UUT descriptions available, the diagnostic reasoning subsystem can be invoked to perform a variety of diagnostic tasks. The diagnostic

reasoning subsystem uses a compiled UUT description to construct and maintain dynamically a belief model about what is properly and improperly functioning as test results become known. This belief model in turn is used to find the best test to make next or recommend the replacement of some module of the UUT.

We currently provide two user interfaces to the diagnostic subsystem. The first is a "mixed initiative" interface intended to be used by a technician as an aide during a diagnostic troubleshooting session. In this mode, FIS is capable of (1) updating the current beliefs about the UUT based on a technician's entry of test results; (2) responding to a technician's query regarding the probability of a fault hypothesis, the merit of a test, the UUT description, or the belief state of FIS; and (3) making suggestions and/or recommendations about the next best action to take (further tests or replacements).

The second mode of use is to use the diagnostic subsystem to produce a traditional test tree by invoking the next best action generator, following its advice (hypothetically), and recursively invoking the next best action generator for each of the possible outcomes of the previously recommended action. In this way, large, complex test trees are generated automatically with no human intervention.

Both the knowledge acquisition subsystem and the diagnostic subsystem of FIS are complex entities that are not easy to describe in a few paragraphs. In the next two major sections, we describe in more detail the motivations behind and the current state of both of these subsystems.

3. KNOWLEDGE ACQUISITION CONSIDERATIONS

In most AI applications, knowledge acquisition is a crucial consideration because it typically has a high cost in work-hours. Fault diagnosis is no exception. To create a computer program that can troubleshoot a piece of equipment with useful accuracy and efficiency, one must invest a considerable amount of effort in knowledge acquisition. The diagnostic reasoning methods chosen must be able to use a UUT description that is easily obtainable by the knowledge engineer and of sufficiently small size. Also, the KAI must allow the fast and accurate transfer of these data to the computer.

3.1. Knowledge Acquisition Trade-offs in Diagnostic Systems

Let us briefly consider some different approaches to achieving some of the diagnostic functions described in Section 1.3 in the light of the relative ease or difficulty of the knowledge required to support them. First, consider the approach of directly writing a test decision tree. This corresponds to the conventional practice of writing programs to control ATE gear in which the burden of diagnostic reasoning lies primarily on the human technician, and what he or she enters into the computer is essentially a decision tree that is to control the ATE autonomously or with minimal human interaction. Although converting such a decision tree into computer code is usually not very labor intensive, much labor is usually involved in producing

the decision tree, because the technician needs to consider many things at each decision point in the tree. The technician must consider what modules are currently suspected of being faulty, in what manner they are suspected, to what degree various modules are certified by previous passed tests, what cost in time or other risk is incurred by performing a certain test, what tests are available to the system, how these tests depend on various faults in various modules, and many other matters. This decision tree approach therefore has high knowledge acquisition cost. It also has the disadvantage of limited fault coverage, inflexibility, lack of explanation capability, and susceptibility to human error and suboptimal decisions.

At another extreme is the approach of entering into the computer a detailed model of the structure of a UUT (e.g., schematic diagrams) and behavior models of the individual components and placing on the computer the burden of deducing enough of the behavior of the replaceable modules and of the whole UUT to do diagnostic reasoning. This approach also requires that the human specify in some way the intended function of the UUT. That is, the diagnosis system must know what part of the UUTs full range of behavior must be tested to certify that the UUT has completely acceptable performance. This approach has the potential of allowing automated diagnosis with low knowledge acquisition cost, since CAD/CAM data could in the future be used to provide the model of the structure of the UUT in computer readable form, and only the intended function (performance specifications) would need to be largely human specified.

Intermediate between these two extremes is the approach of providing to the diagnostic system a simplified model of the qualitative behavior of the replaceable modules and the structure (connectivity) of the UUT, as well as UUT performance specifications. The diagnostic system then can deduce the qualitative relations between test results and possible faulty modules. This then enables it to perform the functions of Section 1.3. Within this paradigm, which is the one adopted by FIS, there are trade-offs. Consider the issue of how detailed a model is needed of each replaceable module. In our current implementation of FIS, we have decided to represent it by a set of *causal rules* describing what signal abnormalities at terminals of the module can be caused by the module failing or by signal abnormalities at other terminals of the module (see Section 3.2). The module's failure statistics are represented by a single number, its failure rate relative to the other modules. We do not require joint statistics of the different ways the module can fail (*failure modes*) because providing them is often infeasible. Indeed, even the module failure rates must often be grossly approximated.

The lack of failure mode statistics makes things difficult for a diagnostic reasoning system. For example, suppose a diagnosis system currently suspects that one of two modules m1 and m2 contains a fault. Then it might be able to determine which is likely to be faulty by finding a test dependent on m1 but not on m2. If the test fails, m1 is faulty, and if the test passes, m1 is less suspect than it was, and m2 more suspect. However, if a human interpreted that passed test, he or she would think about what failure mode of m1 tested okay. If it were the same failure mode that made m1 initially suspect, m1 would be exonerated. If the passed test depended on a completely independent failure mode of m1, then m1 would be only slightly certified.

Thus, a diagnosis system could benefit from knowing the complete joint statistics of the failure modes of each module and the qualitative relation between these and measurable abnormalities at the terminals of the module. However, since providing such data is often infeasible, we have considered and implemented various approximate methods for handling the certification of modules after passed tests without using failure mode statistics (see Section 4.4).

3.2. The UUT Description Used in FIS

The knowledge required to describe a specific UUT type includes the following:

(a) A graphics description for display of a block diagram of the UUT.
(b) A list of module (component) names.
(c) Causal rules for each module.
(d) Connectivity description.
(e) Module replacement costs.
(f) Relative a priori module and UUT failure probabilities.
(g) A list of performance tests for the UUT.
(h) A list of additional tests available for fault isolation.
(i) Test cost rule set.

The graphics description (a) is generated automatically by FIS from the connectivity description (d) and consists of a block diagram with the modules labeled by their names and current fault probabilities. The user then edits this diagram to optimize its appearance.

The most interesting UUT-descriptive knowledge is (c) above. It consists of causal rules relating measurable (at least in principle) parameters among various terminals of each module. These are intended to describe the behavior of each module in its context. Thus, where loading and other effects of Kirchhoff's laws affect behavior, these are assumed taken into account. We are not restricted to unidirectional modules with fixed input–output voltage behavior. For the common case in which this behavior is independent of context, the library module descriptions, when available, can be invoked without modification. These rules contain both information about the correct behavior of UUT modules and about fault modes of modules, as discussed later. The rules are of the following two forms, called through rules and fault rules, respectively:

1. Given *precondition, parameter1 abnormality1* at *terminal1* can cause *parameter2 abnormality2* at *terminal2.*
2. Faulty *modulename* can cause *parameter abnormality* at *terminal.*

Here a precondition is a predicate on the state of the UUT inputs (or input history). Preconditions are optional and are used primarily to model devices such as mul-

tiplexers and devices with enable inputs in which a causal path is present or absent depending on some signal conditions. Although we have experimented with elaborate precondition schemes involving estimating probabilities of control signal values, we have found it quite adequate simply to refer to the stimulus data associated with the current test to see if the input stimuli correspond to those named in a precondition. An abnormality is a qualitative value such as high or low. For example, a rule of type 1 is "Given that power supply 3 is in the on state, DC voltage high at t3 can cause the frequency to be low at t4," in English paraphrase. This might describe the way a voltage-controlled oscillator propagates a voltage error, given that its power supply voltage is in spec. In the case of a module with two inputs both affecting the same output parameter, such as a summing junction, a problem can arise. Suppose two of the rules are "input1 DC high can cause output DC high" and "input2 DC high can cause output DC high." Then, if "output DC high" is suspected, both inputs are to be suspected high. However, one might be very high and the other slightly low, and the latter would be missed. However, if such a case occurs, the system will find a fault leading to the high input, and the ensuing repair may even cure the slightly low input. Otherwise, a second diagnostic pass can search for it. Each causal rule is associated with the module immediately connected to the terminals appearing in the rule.

Rules of type 1 represent the correct behavior of a module in context in the form of qualitative relations among quantities at its terminals. Rule type 2 represents limited knowledge about how a given module can fail. Note that even if we had no such fault model knowledge available, we could allow the system to include all conceivable rules of type 2 by default and still have a viable diagnosis system. That is, we could assume that any abnormality at a terminal could have been caused by the failure of any directly connected module. This would lead to somewhat larger ambiguity sets (suspect sets) than with a more minimal set of type 2 rules, but most of the diagnostic power of FIS comes from rules of type 1, since it is largely by chaining them that we determine what portion of the UUT affects a given measurement. When in doubt about the inclusion of any rule, we must include it or else we risk missing some faults.

The connectivity description [item (d) above] is given in terms of triples (module1 terminal module2). Note that in the connectivity description there are always two modules connected to one terminal. This is because metal conductors are modeled as modules themselves to capture their open-circuit fault behavior. Thus, three module terminals connected together in a Y become three UUT terminals connecting six module terminals as shown in Figure 3. The connectivity knowledge is also represented redundantly in the causal rules, which refer to terminals and are each associated with a module.

Item (e), the module replacement costs, are used to recommend the replacement of a module when that action minimizes the expected total cost of testing and replacement (see Section 4.6). These costs are manifested primarily in time and risk of equipment damage and are represented in time units. Item (f) is to be obtained ideally from statistics of actual failures of each module in the current UUT type. If that is unavailable, failure rates for the most specific generic class

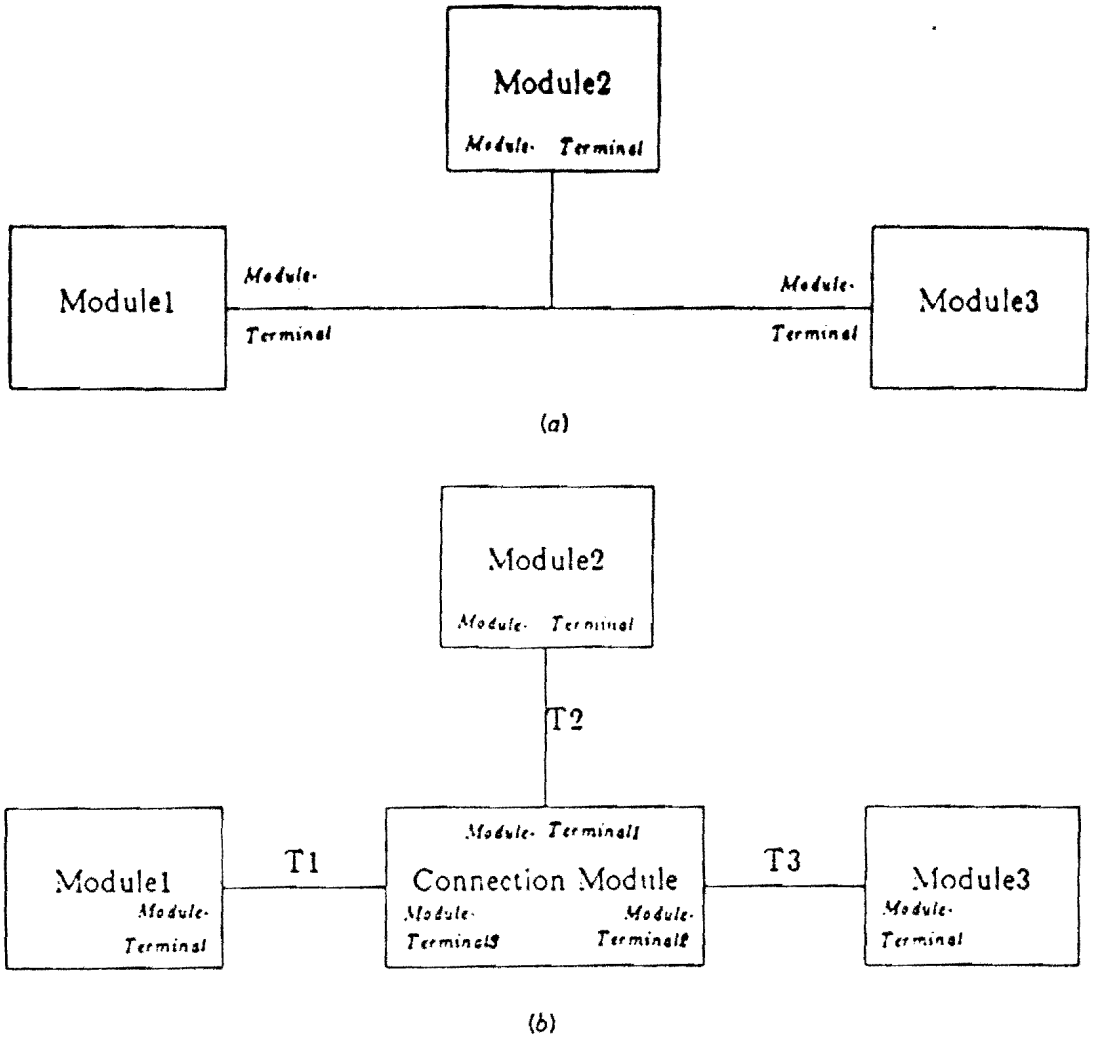


Figure 3. (a) Conventional drawing of a connection and (b) our model of a connection.

to which a module belongs and for which statistics are available should be used, with equal probabilities the last resort. The a priori UUT failure probability is simply the estimated probability that the UUT has any faulty module. Item (g) is a list of tests that, if all were made and passed, would imply that the UUT is for practical purposes certified as fully functional. Item (h) is a list of tests that are potentially useful for fault isolation but that are not needed to certify the UUT as fully functional.

A test of either type consists of several items of data, as illustrated in Figure 4 by the small sample of tests from a radar UUT. Each has a unique test number, a terminal to use as test point, a parameter to be measured, a set of qualitative test outcomes and the quantitative specifications defining them, the units of measurement of the specifications, the test type (performance or diagnostic only), and cost-code data. In the case of analog parameters such as frequency, the specifications would be numerical ranges. In the digital case, they would be values of bits or words. The test cost code gives a prescribed state or mode of use for each of the stimulus or measurement instruments or other hardware involved in each test. z is the "don't care" condition and 1 is the "off" state. Item (i) is a small and simple

Test#	Terminal	Parameter	Qualvals & Specs	Units	Type
t0300a	outbus	d-reply	(ok (ecl-1) bad (ecl-0))	logic	p
t0300b	outbus	ovrtemp	(ok (ecl-1) bad (ecl-0))	logic	p
t0300c	outbus	reply	(ok (ecl-1) bad (ecl-0))	logic	p
t0302a	outbus	d-reply	(ok (ecl-0) bad (ecl-1))	logic	p
t0302b	outbus	ovrtemp	(ok (ecl-1) bad (ecl-0))	logic	p
t0302c	outbus	reply	(ok (ecl-0) bad (ecl-1))	logic	p
t0386	outbus	agcpar	(ok (ecl-1) bad (ecl-0))	logic	p
t0388	outbus	agcpar	(ok (ecl-0) bad (ecl-1))	logic	p
t0390	outbus	sdipar	(ok (ecl-0) bad (ecl-1))	logic	p
t0392	j9bus	g-off-par	(ok (ttl-1) bad (ttl-0))	logic	p
t0394	outbus	sdipar	(ok (ecl-1) bad (ecl-0))	logic	p

Test#	Test Cost-Code Information												
	m	p	d	sc	pw	fq	am	fm	anl	tmr	dvm	arr	dd
t0300a	z	1	1	z	z	z	z	z	z	z	z	z	z
t0300b	z	1	1	z	z	z	z	z	z	z	z	z	z
t0300c	z	1	2	z	z	z	z	z	z	z	z	z	z
t0302a	z	1	2	z	z	z	z	z	z	z	z	z	z
t0302b	z	1	2	z	z	z	z	z	z	z	z	z	z
t0302c	z	1	3	z	z	z	z	z	z	z	z	z	z
t0386	z	1	6	z	z	z	z	z	z	z	z	z	z
t0388	z	1	7	z	z	z	z	z	z	z	z	z	z
t0390	z	1	10	z	z	z	z	z	z	z	z	z	z
t0392	z	1	11	z	z	z	z	z	z	z	z	z	z
t0394	z	1	12	z	z	z	z	z	z	z	z	z	z

Figure 4. Description of available tests.

set of rules giving the cost (in time units) of using each test instrument in a current test given the previously made test. For example, if the timer (tmr) were in the "off" state (tmr = 1) in one test and in state 23 in the next test, we would capture the fact that the timer needs a 50-second warm-up time with the rule "(timer 1 to x) implies cost = 50." This compact approach was motivated by the fact that there are n^2 possible transitions from one of n tests to a second one, and therefore n^2 costs, a prohibitive amount of data to be acquired explicitly.

3.3. Knowledge Acquisition Strategies

The description of a new UUT is developed by a user with the aid of the KAI in Figure 2. In the future, it may be possible to replace the user at least partially with a CAD/CAM database describing the UUT. For the most part, the UUT descriptive data described in Section 3.2 are acquired in a direct manner from the user via a menu system in the current implementation of FIS. The largest data items are the lists of tests and the causal rules of the modules. For the latter, the user needs to transfer to the computer his or her understanding of each module's behavior. This requires a large number of causal rules for a typical UUT (30 rules per module is typical). Therefore, efficient transfer of knowledge is important. For this reason, we have experimented with several means of specifying causal rules for a module:

1. Entering individual causal rules.
2. Entering several causal rules at once by expressing them in a compact form; for example, "The DC voltage at all terminals of module3 can cause the failure of any parameter at terminal6," in English paraphrase.
3. Obtaining a complete rule set of a familiar module from the library by specifying the UUT-specific module name, the generic module name, and the associations between UUT terminal names and library module terminal names.
4. Modifying the current rule set for a module of the UUT by naming properties (partial descriptions of a module rule set) from the library and giving the associations between UUT terminal names and property terminal names. This leads to both additions and deletions of rules.

Item 2 can take advantage of various patterns in the rule set of a module. The example given causes the generation of all combinations (the "outer product") of the given causes and the given effects. Such notation is useful for a complex module for which any abnormality at one terminal can, under some circumstances, cause any abnormality at some second terminal, or for which the user has an incomplete understanding. Another example is "All abnormalities at terminal5 of module3 can cause the failure of the same parameter at terminal6," in English paraphrase. This yields a one-to-one correspondence between causes and effects. Various other compact notations have been tried. The KAI can expand these immediately to yield individual rules that can be edited further, if desired, by the user.

Every rule for each UUT module carries a label of "mandatory" or "forbidden" for use during editing. After compilation, only the mandatory ones survive and the label is dropped. The reason for this feature is evident from item 4. Rules obtained by invoking a property are of both the mandatory and forbidden types, whereas all rules obtained by method 1, 2, or 3 are mandatory. Thus, when a new rule is obtained by any of the four methods, if it is mandatory and not present in the UUT module rule set, it will be added. If it is mandatory but already present and marked mandatory in the UUT module rule set, nothing is changed. If it is mandatory and present but marked forbidden in the UUT module rule set, then the user is called

on to resolve the conflict. The case of dealing with a forbidden new rule is handled exactly symmetrically. Thus, when there is a conflict, the user resolves it, and when there is not, the new rule is "unioned" into the UUT module rule set.

Besides speed of acquisition, in the case of the causal rules it was found useful to assist the user in ensuring that the rule set is a good one. This means that the rule set has upstream causal paths from each possible failed test result to a set of modules that is sure to contain a faulty one. At the same time, each such ambiguity set should not contain any module that we know could not cause the test result. To facilitate enforcing this, FIS can sort the rule set by effects (right-hand sides) and give the causes of all the rules having each effect. Figure 5 shows a sample of such a list. This enables us to follow the causal paths upstream quickly by eye to evaluate the rule set. FIS also can display the computed ambiguity sets for each possible test. This is a useful tool in validating the causal rule set.

3.4. The Knowledge Compiler

The UUT descriptive data described in Section 3.2 are processed by the knowledge compiler to produce data structures allowing fast diagnosis time processing. For example, the *ambiguity sets* of each test are computed and stored; that is, for each abnormal qualitative value of the test, the causal rules associated with the terminal and parameter tested are obtained. The right-hand sides (effects) are matched against the failed test result. The left-hand (causes) sides of the matching rules now represent possible causes of the failed test. These are now matched recursively against rules until all culpable modules are found. Also, for each of these modules, those rules having that module as cause and having an effect on the causal path to the test result are recorded. These are called *immediate effects* and are used in performing module certification after a passed test (see Section 4.4).

The compiler also sorts the set of tests in several ways so that we can quickly look up the set of available tests at a given terminal, measuring a given parameter, dependent on a given module via a given fault rule, and so on. These data structures will be used by the best test heuristics (Section 4.6).

4. DIAGNOSTIC REASONING

The diagnostic functions of FIS are illustrated in Figure 6 by the menu FIS displays when ready to be used in the technician's aid mode. The major functions are *make-test* and *best-test*, which we will describe in detail. Also of interest is *forget-last*, which allows the user to retract the last make-test he or she performed. This is useful for considering hypothetical sequences of tests and test results.

4.1. Beliefs About Faults in the UUT

The updating of beliefs about faults after a test is a crucial function in a diagnosis system. Poor selection of tests can delay the diagnosis, but inaccurate use of their

Effect	Cause	Precondition
(agcpar logic b)	adc	t
(bit-rf freq hi)	(lo-out freq hi)	t
	(ifosc2 freq hi)	t
(bit-rf freq lo)	(lo-out freq lo)	t
	(ifosc2 freq lo)	t
	(td-cnt any b)	t
(bit-rf pulsespread hi)	tdrive	t
	(td-cnt any b)	t
(bit-rf pulsespread lo)	(td-cnt any b)	t
	tdrive	t
(bit-rf pwr hi)	(td-cnt any b)	t
	tdrive	t
(bit-rf pwr lo)	tdrive	t
	(lo-out pwr lo)	t
	(td-cnt any b)	t
	(ifosc2 pwr lo)	t
(cagc1 logic b)	adc	t
(cagc2 logic b)	adc	t
(cagc3 logic b)	adc	t
(first-lo freq hi)	l-osc	t
	(locnt any b)	t
	(refout freq hi)	t
(first-lo freq lo)	l-osc	t
	(locnt any b)	t
	(refout freq lo)	t
(first-lo pwr lo)	l-osc	t
(fmr dc hi)	pwr&cnt	t
(fmr dc lo)	pwr&cnt	t
(fmr noise hi)	pwr&cnt	t

Figure 5. Sorting the UUT rules by effects.

results can cause erroneous diagnosis. When a test is made in FIS (see Sections 4.2–4.4), the result is processed along with the compiled UUT description and the current beliefs, and produces as output updated beliefs. This deterministic data can then be processed (see Section 4.5) to obtain probabilities of fault hypotheses. The beliefs consist of five types of data and represent what FIS currently knows about the state of the UUT:

INITIALIZATION	INSPECTION	TROUBLESHOOTING	CONTROL
	sh: show-history	rc: repeat-command	li: lisp
	al: active-list	bt: best-test	cu: change-uut
	du: draw-uut	fa: forget-all	hu: hardcopy
	tt: trace-tpt	fl: forget-last	qt: quit
	ut: untrace-tpt	mt: make-test	un: unix
	ct: console-trace		
	gt: graphic-trace		
	ft: file-trace		
	tn: trace-net		
	as: amb-set		
	ss: show-specs		
	sp: show-probs		
	w: window		
Enter command or ? -->			

Figure 6. The menu appearing when FIS is used as a technician's aid.

1. Quantitative and qualitative values of tests made.
2. The set of current *ambiguity sets*, that is, the sets of modules each of which is believed to contain at least one faulty module. This accrues one new ambiguity set per failed test.
3. The current setup of the UUT–ATE system, represented by the *test cost code*, corresponding to the last test made (see Section 3.2). This is needed for comparison with the test cost code of the following test to compute its cost.
4. Certification numbers for each immediate effect of each module. These indicate the number of passed tests dependent on the given module via the fault rule whose cause is the module and whose effect is the *immediate effect*.
5. The certification factors to be multiplied by the a priori relative failure rates a_i of the modules when calculating fault probabilities (see Section 4.5). These are initially 1.0, indicating no evidence from tests that module i is good. Complete certification is indicated by 0.0. These factors are computed from the certification numbers as described in Section 4.4.

The five types of belief information described in this section together with the a priori module failure probabilities include all the important information about the possible faults in the UUT at a given time. However, this information is most useful both to a technician and to the routine that computes the next best test if it is converted into the form of probabilities of fault hypotheses, as described in Section 4.5.

4.2. Making a Test

When a technician (or ATE) makes a test on a UUT, the result is reported to FIS as a quantitative or qualitative value of the parameter measured. Figure 7 illustrates

Enter command or ? --> **mt**

Enter terminal at which this test was made--> **rf-out**

TEST	PARAMETERS
rf-out	pwr
	pulsespread
	noise
	freq

Enter parameter which was measured--> **pwr**

TEST	SETUPS
rf-out	52-1-7
pwr	51-2-3
	20-21-1-1

Enter setup under which this test was made --> **52-1-7**

TEST	QUALVAL	SPEC
rf-out	ok	((-inf -65))
pwr	hi	((-65 inf))
52-1-7		

Enter measured value or qualval --> **hi**

Figure 7. Making a test in FIS.

the process of making a test in the technician's aid mode. A test can be specified by naming the terminal, parameter, and setup (part of the test cost code) of the desired test, as shown, or by naming the test number. If a quantitative value is entered, FIS converts this to one of the qualitative values for that test, such as high, low, or okay for an analog measurement, by comparing it with the specified values. In processing the test result, FIS first makes a new copy (using constructive LISP operations for compactness) of the current UUT state so that it can undo tests if desired. Then it updates the test cost code to the one reported with the test. Failed and passed tests are processed differently.

4.3. Processing Failed Tests

If the specification is violated, the ambiguity set for the particular qualitative value (such as low or high) of the particular test taken is found. This set is then appended

to the set of ambiguity sets. The ambiguity set is found by lookup, since all such sets are precomputed by the knowledge compiler. In previous versions of FIS, the ambiguity sets were computed when a test was made. This was more time consuming and was done because various deductions about qualitative values of electrical parameters were made while processing a test result [2], which required following the rules at diagnosis time. However, we have decided to abandon these deductions because they are marginally useful. The important information gleaned from a set of failed tests is the corresponding set of ambiguity sets.

The simple cascade network of Figure 8 illustrates the small size of ambiguity sets computed with FIS. Note, for example, that if a test shows the DC voltage to be high at T6, only the highpass filter can be the cause. The reader can readily see how the ambiguity sets were found by mentally chaining through the given rules from right to left from a test result.

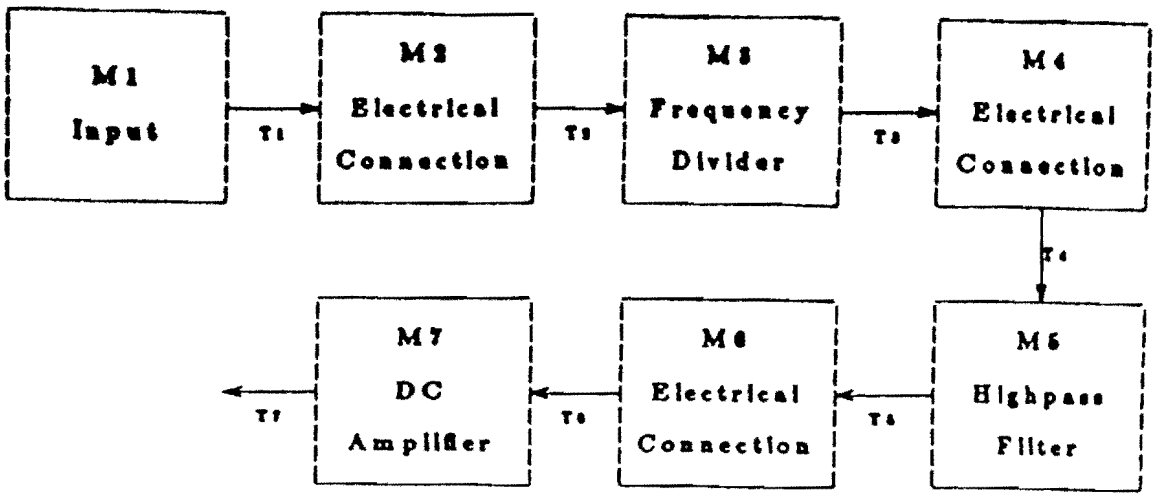
Having added the ambiguity set of the current failed test to the UUT beliefs, FIS now proceeds with the probabilistic computations described in Section 4.5, as indicated in Section 3.1.

4.4. Processing Passed Tests

In the case of a passed test (qualitative value okay), FIS first notes for each module on which the test depends how many of the abnormalities that it can immediately cause (at its terminals) lie on the causal path to the failed (e.g., low or high) outcomes of the test. It finds these *immediate effects* by direct lookup of precompiled *ambiguity sets* for the test (see Section 3.4). Then it must determine how much to reduce the suspicion (fault probability) of the modules involved.

As indicated in Section 3.1, the manner in which a passed test results in reducing the suspicion of a module needs careful consideration. We have experimented with several schemes for certification. In all cases, we convert the information gained from the various tests into a single certification factor for each module, as described in Section 4.1. In earlier versions of FIS we tried a simple linear scheme. For example, if a module has ten *fault rules* and four of the rules have right-hand sides on the causal path to the test, then the certification factor for the module is $1 - \frac{4}{10} = 0.6$. This rough approximation was used because the a priori statistics of the various modes of failure of a given module are seldom available. Thus, the a priori fault probability of the module would be multiplied by 0.6 before being used in the probability calculations of Section 4.5. However, two problems were found with this. First, the different ways a module can fail are often not mutually exclusive. Thus, the first passed test should have a greater effect than later ones. Second, if a module is currently strongly suspected of being faulty because one or more tests depending on it failed, then it should be certified more strongly by some passed tests than by others (see Section 3.1).

To alleviate these difficulties while still avoiding requiring joint statistics data for the failure modes of a module, we are currently experimenting with the following scheme. We keep track of how many of the tests that depend on a given failure mode of a module have been made and passed. Then we take the sum s of these



Test Result	Ambiguity Set
T6 freq high	M1, M3
T6 dc low	M5, M6, M7
T6 dc hi	M5

Rules Fired in Propagation of above Test Results

Cause	Effect	Comments
T5 freq high	T6 freq high	
T4 freq high	T5 freq high	
T3 freq high	T4 freq high	
T2 freq high	T3 freq high	
M3	T3 freq high	M3 not dividing
T1 freq high	T2 freq high	
M1	T1 freq high	M1 has high frequency
T5 dc low	T6 dc low	
M6	T6 dc low	M6 open, M7 pulls low
M7	T6 dc low	M7 input grounded
M5	T6 dc low	M5 output has offset
M5	T6 dc high	M5 output has offset
T5 dc high	T6 dc high	

Figure 8. Simple examples of the propagation of a test result.

and compute exp ($-ks$) as the certification factor for the module, with the constant k chosen empirically to optimize performance. This addresses the first problem, the requirement that successive passed tests reduce the fault probabilities by progressively lesser amounts. Now if the module in question is contained in any ambiguity set then exp ($-ks - k_a s_a$) is taken as the certification factor, where s_a is the number of passed tests that depend on those particular immediate effects of the module that are responsible for that module being in an ambiguity set. The constant k_a is chosen empirically to optimize performance. This addresses the second problem mentioned above.

4.5. Calculating the Fault Probabilities

In problem domains that are complex and for which deep models relating structure and function are incomplete, such as medical diagnosis, it is often impossible to do evidential reasoning in a precise manner, since the relevant joint statistics are unavailable. Therefore, approximate methods have been found useful such as those in MYCIN [16] and PROSPECTOR [17] and the Dempster-Shafer [18] formalism. However, in the domain of diagnosis of human-engineered systems constructed from discrete modules, it is possible to use more precise probabilistic methods exploiting the regularities of this domain. One regularity is that the information gleaned from a failed test can often be described as an ambiguity set, a set within which at least one faulty module must lie. Another is that it is often an acceptable approximation to assume that the replaceable modules fail independently of one another with their own a priori probabilities.

A fault hypothesis is defined here as a Boolean combination of assertions that individual modules are good or faulty. For example, $x_1 \,\&\, \overline{x_2} \,\&\, x_3$ is the hypothesis that modules 1 and 3 are bad and module 2 is good, without regard for the other modules. FIS contains a program that computes the probability of a fault hypothesis in light of the test results to date. We make the assumption that the individual module fault propositions x_i are statistically independent of each other with relative a priori probabilities a_i, so that $a_i a_j$ is the relative a priori probability of $x_i \,\&\, x_j$, for example. We are thus ignoring the case of one component failing first and stressing a second component so that it fails also. This will cause some error in probability calculations, which will affect primarily the selection of tests, and therefore the cost (e.g., time and money) of diagnosis, and not its accuracy. Barring this coupling, we can view double faults as independent events—the second of which occurred by chance in the time between the occurrence of the first and the time of testing.

For clarity, we will first describe how the program processes the results of failed test results only. Then we will treat the case of some failed and some passed tests and, finally, the case of all passed tests. As described in Section 4.3, each failed test gives rise to an ambiguity set containing all modules that could have caused that abnormal measurement. The set of these ambiguity sets corresponds to a Boolean conjunctive normal form expression in the module fault propositions x_i: at least one member of the first ambiguity set is faulty and at least one member of the second ambiguity set is faulty, and so on. Let T denote this expression and H denote an arbitrary Boolean function of the x_i and their complements, describing a hypothesis whose current probability is desired. From Bayes' rule the current probability of H is given by

$$P(H|T) = \frac{P(H \,\&\, T)}{P(T)}. \tag{1}$$

$P(B)$ is defined for an arbitrary Boolean function B as the a priori probability of the proposition B, that is, the a priori probability that the fault state of the UUT is consistent with B.

The problem is now reduced to that of computing $P(B)$ for an arbitrary Boolean function of the literals x_i. The strategy is to manipulate B so that all terms of conjunctions are statistically independent (involve no common x_i) and all terms of disjunctions are nonoverlapping (mutually exclusive). We can then replace the x_i with the corresponding a priori component failure probabilities a_i and the three Boolean operations with arithmetic operations using the following three laws from probability theory:

(A) If $P(X) = p$, then $P(\bar{X}) = 1 - p$.
(B) If $P(X) = p$ and $P(Y) = q$, then $P(X \vee Y) = p + q$ iff $X \& Y = 0$.
(C) If $P(X) = p$, $P(Y) = q$, and X and Y are statistically independent, then $P(X \& Y) = p \times q$, prior to the use of a_i in the above paragraph.

We have developed an efficient algorithm for performing the computation of Equation (1) outlined above. For those interested we give a concise trace of its behavior on an example problem. Suppose that three tests have failed and our current ambiguity sets are $\{m_2, m_3, m_4\}$, $\{m_4, m_5, m_6\}$, and $\{m_7\}$. Let us compute the probability of the hypothesis x_7 that module m_7 is faulty. The ambiguity sets overlap interestingly (at least two faults are present) and make a good illustration. We define the shorthand notation $\overline{P(B)} = 1 - P(B)$ and x_i is denoted by i. Then the numerator of Equation (1) becomes

1. $P(H \& T) = P[(2 \vee 3 \vee 4) \& (4 \vee 5 \vee 6) \& (6 \vee 7) \& 7]$;
2. $= P[(2 \vee 3 \vee 4) \& (4 \vee 5 \vee 6) \& 7]$ Boolean absorption;
3. $= P[\overline{(\bar{2} \& \bar{3} \& \bar{4}) \vee (\bar{4} \& \bar{5} \& \bar{6}) \vee \bar{7}}]$ DeMorgan's law;
4. $= \bar{P}[(\bar{2} \& \bar{3} \& \bar{4}) \vee (\bar{4} \& \bar{5} \& \bar{6}) \vee \bar{7}]$ law (A) above;
5. $= \bar{P}[(\bar{2} \& \bar{3} \& \bar{4} \& (5 \vee 6) \& 7) \vee (\bar{4} \& \bar{5} \& \bar{6} \& 7) \vee \bar{7}]$ disjunction overlap removal;
6. $= 1 - P(\bar{2} \& \bar{3} \& \bar{4} \& (5 \vee 6) \& 7) - P(\bar{4} \& \bar{5} \& \bar{6} \& 7) - P(\bar{7})$ law (B);
7. $= 1 - P(\bar{2})\,P(\bar{3})\,P(\bar{4})\,P(5 \vee 6)\,P(7) - P(\bar{4})\,P(\bar{5})\,P(\bar{6})\,P(7) - P(\bar{7})$ law (C);
8. $= 1 - \bar{P}(2)\,\bar{P}(3)\,\bar{P}(4)\,P(5 \vee 6)\,P(7) - \bar{P}(4)\,\bar{P}(5)\,\bar{P}(6)\,P(7) - \bar{P}(7)$ law (A)
9. $P(5 \vee 6)$ becomes $P(\overline{\bar{5} \& \bar{6}})$ DeMorgan's law;
10. $P(H \& T) = 1 - \bar{P}(2)\,\bar{P}(3)\,\bar{P}(4)\,(\overline{\bar{P}(5)\,\bar{P}(6)})\,P(7) - \bar{P}(4)\,\bar{P}(5)\,\bar{P}(6)\,P(7) - \bar{P}(7)$.

The $P(i)$ are simply the a_i. The denominator of Equation (1) is treated similarly.

The above method is used to compute the current fault probabilities of the individual modules after each test. It can also be invoked by the user to compute the probability of any fault hypothesis, even one including some fault states inconsistent with T.

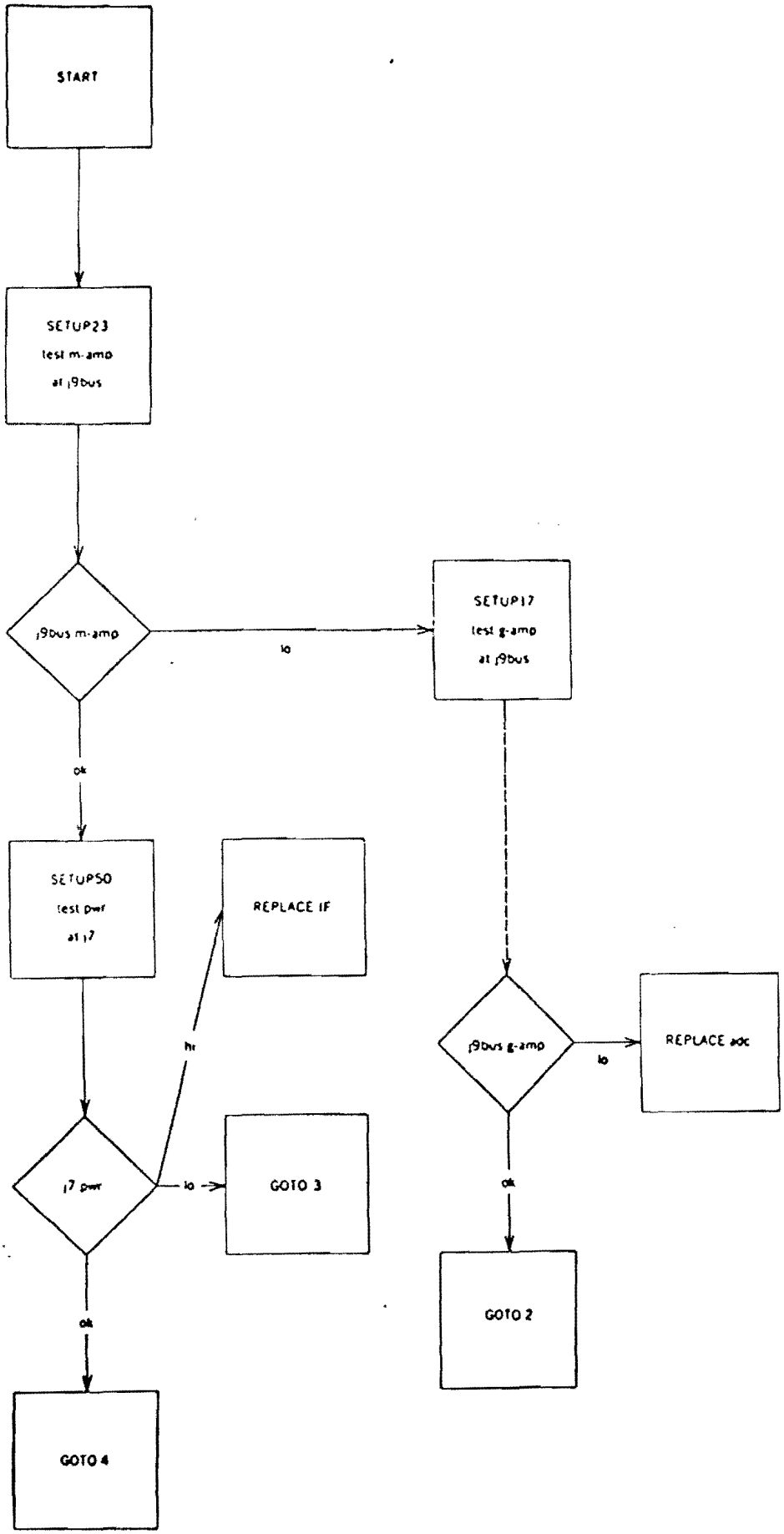
Now we will treat the case of having at least one passed test and at least one failed test. As described in Section 4.4, for each module m_i, FIS computes a certification factor c_i that represents the evidence from passed tests. Thus, module m_i is regarded as certified by the amount c_i and the a priori failure rate a_i is simply replaced with $c_i a_i$ in the probability calculations above.

Finally, if all tests have passed so far, then there are no ambiguity sets and the probability algorithm above is not used. Instead the a priori module fault probabilities are renormalized using the a priori UUT fault probability P_0 and the certified a priori module probabilities $c_i a_i$.

4.6. Next Best Test and Replacement Decisions

The problem of finding the "best" test to make next in troubleshooting is analogous to that of playing a game against an opponent who responds randomly. The test result is the opponent's move and the test selection is our move. The object of our side is to minimize the cost of finding the faults. We could think of the opponent as confounding our efforts by giving test results that sometimes further our goal and sometimes do not. Applying a mini-average algorithm to the "game tree" [13] yields an optimal solution but is infeasible for large branching factors. In FIS, selecting a test amounts to choosing three things—setup, test point, and parameter measured. Thus, the number of available tests can be large. At the next level of the game tree, there can be several test results, one for each of its qualitative values, for example, low, high, and okay. We take the approach of first heuristically screening the set of possible tests to reduce their number significantly and then examining only one level of the game tree to pick the best of the remaining candidate tests. Specifically, we find the screened test that maximizes the ratio of the expected information gain to the estimated test cost. The expected information gain is a weighted average of an entropy-based evaluation function over the several possible test results, where the weights are the estimated test result probabilities.

In earlier versions, we screened the tests by simply exhaustively computing the initial (opening move) evaluation function values at compilation time and then taking the best n of these as our screened candidate tests during diagnosis. This is not very satisfactory, however, since the initial ordering of the tests by their merit is often very different from their ordering after one or more tests have been made. Therefore, we are currently experimenting with heuristic screening techniques that are highly dependent on the current UUT beliefs. This is expected to result in a small set of high-quality candidates, allowing rapid evaluation by the entropy–cost routine. One source of good heuristics is the observation that tests dependent on some but not all of the highly suspect modules are powerful. Another is that tests made under the same UUT stimulus conditions as some failed test are more likely to fail and thus contribute significantly to fault isolation. Tests that have low cost are to be preferred to tests of otherwise equal merit but with higher cost. If


START
SETUP23
test m-amp
at j9bus
j9bus m-amp
lo
SETUP17
test g-amp
at j9bus
ok
SETUP50
test pwr
at j7
REPLACE IF
hi
j9bus g-amp
lo
REPLACE adc
j7 pwr
lo
GOTO 3
ok
GOTO 2
ok
GOTO 4

the test tree. Each test or replacement recommendation represents a node in the test tree. Figure 9 shows a sample of output from the automatic test tree generator.

5. CURRENT APPLICATIONS OF FIS

We have been testing and refining FIS on real analog UUTs to assess its performance and practicality and to provide ideas for improvements. We have been modeling a radar receiver–exciter subsystem with the goal of generating test trees (called diagnostic and functional flowcharts) that are comparable in quality to those written by test programmers. As a second application, we have recently begun applying FIS to a Navy sonar system as a technician's aid.

In the radar application, the goal is to demonstrate that FIS can relieve the human test programmer of part of the labor of producing a *test program set* (TPS), which is a program that controls ATE gear in the automatic execution of a test tree. It is the automatic test tree generation capability of FIS (see Section 4.7) that is used here. The part of the human effort that is relieved is the sequencing of the possible tests and the determination of when some modules warrant replacement and which ones. In exchange for this, FIS places a modest knowledge acquisition burden on the user, primarily in the form of causal rules and test cost information. Other parts of this task that we leave under human control are the choice of test equipment and the determination of what constitutes a sufficient set of tests to certify that the UUT is performing correctly and to do fault isolation. Figure 10 shows a hardcopy version of the more detailed CRT color graphics display of the radar unit. This is useful during the development of the UUT description, when the technician's aid mode is invoked to test interactively the diagnostic performance before test trees are generated. We are currently making refinements in FIS to make it practical for the TPS application. This includes primarily the improvement of the speed and accuracy of the best-test computation and the more intelligent handling of passed tests.

The second application involves the use of FIS as a maintenance advisor to a technician. It will be installed in a rugged portable computer and will direct or assist a technician in the troubleshooting of a complex, primarily analog sonar subsystem consisting of 105 replaceable modules, using manual test equipment. The primary functions will be the recommendation of a next best-test and the reporting of the FIS beliefs after a test is made.

The applications brought into focus the issue of computational complexity, since they require speedy operation even for large UUTs. The computational complexity of the propagation of test results through the causal rules appears to be approximately linear in the number of terminals in the UUT, since during the finding of an ambiguity set the number of rules invoked at each terminal depends only on the complexity of the immediately connected modules, not on the size of the UUT, and each terminal is visited at most once for each parameter. However, this is done at compile time, and so it need not be fast. In both technician's aid applications and test tree generation applications, most of the computation consists of make-

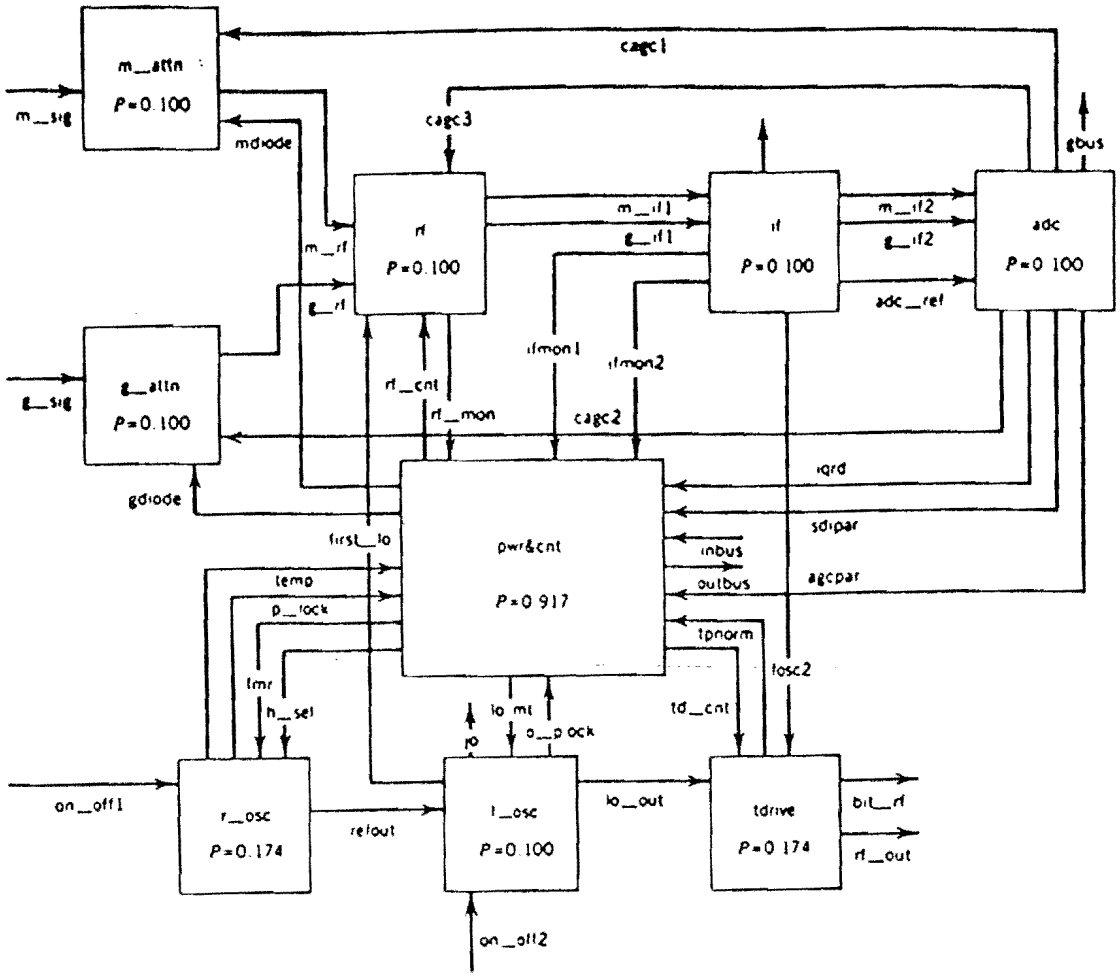


Figure 10. A model of a radar receiver/exciter in FIS.

test operations and best-test operations. The latter is the more computationally costly. The best-test selection procedure is too complex to perform a rigorous complexity analysis easily, but its complexity has two major contributions—one from the heuristic screening of available tests and one from the entropy calculations on the remaining candidates. The latter currently takes on the order of a few seconds for typical UUTs but is expected to be improved considerably by implementation changes. Since this is invoked once per candidate test, it is important that the heuristic screening procedures produce a small number of candidates. We expect that the heuristics described in Section 4.6 will lead to no more than ten candidate tests when they are fully implemented. The running time for the heuristic procedures is expected to be less than that of the entropy calculations.

6. CONCLUSIONS

The FIS project has resulted in an expert system shell that can efficiently acquire a causal model of the behavior of a UUT adequate for diagnostic reasoning and use that model in a variety of diagnostic settings to recommend tests and replacements and to interpret test results. Even with relatively small UUTs, the qualitative

causal modeling of components clearly shows its power in the correctness and small size of the ambiguity sets resulting from each test, as compared to a topologically based troubleshooting system such as those of Cantone et al. [13] and Simpson and Balaban [14]. In the example of Figure 8, a purely topological troubleshooting system would miss the loading problem of M7 at T6 and suspect all the upstream modules for any failed test. There is no known nontopological system with which we can directly compare FIS.

The causal model required by FIS is modest in size and in our experience can be generated efficiently using the methods of Section 3.3. This suggests that our approach will allow efficient knowledge acquisition and accurate diagnosis for larger UUTs.

We plan to continue refining the FIS prototype, using lessons learned from the radar and sonar applications. Most of the additional improvements are expected to be in the user interface, for example, to make the graphics more suitable for large UUTs in the technician's aid application and to improve the efficiency of knowledge acquisition. We will also further refine the best test calculations (Section 4.6) to achieve as much speed and accuracy as possible and make other improvements to facilitate the transition of this technology into further development and eventual use in the field.

REFERENCES

1. F. Pipitone, An Expert System for Electronics Troubleshooting Based on Function and Connectivity. In *Proceedings of the IEEE First Conference on Artificial Intelligence Applications*, pp. 133–138, IEEE, Washington, D.C., 1984.
2. F. Pipitone, The FIS Electronics Troubleshooting System, *IEEE Computer Magazine*, pp. 68–76, July 1986.
3. J. DeKleer and B. C. Williams, Diagnosing Multiple Faults. In *Artificial Intelligence*, North-Holland, Amsterdam, 1987.
4. H. Geffner and J. Pearl, Distributed Diagnosis of Systems with Multiple Faults. Technical Report R-66 CSD-8600, UCLA Computer Science Department, Los Angeles, 1986.
5. J. Pearl, Distributed Revision of Belief Commitment in Multi-Hypotheses Interpretation, In *Proceedings 2nd AAAI Workshop on Uncertainty in Artificial Intelligence*, pp. 201–209, AAAI, Menlo Park, CA, 1986.
6. M. R. Genesereth, Diagnosis Using Hierarchical Design Models. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 278–283, AAAI, Menlo Park, CA, 1982.
7. R. Davis, H. Schrobe, W. Hamscher, K. Wieckert, M. Shirley, and S. Polit, Diagnosis Based on Description of Structure and Function. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 137–142, AAAI, Menlo Park, CA, 1982.
8. J. S. Brown, R. Burton, and J. DeKleer, Pedagogical, Natural Language and Knowledge Engineering Techniques in SOPHIE I, II, and III. In *Intelligent Tutoring Systems*, D. Sleeman and J. S. Brown (eds.), pp. 227–282, Academic Press, London, 1982.

9. R. M. Stallman and G. J. Sussman, Problem Solving About Electrical Circuits. In *Artificial Intelligence: An MIT Perspective*, P. H. Winston and R. H. Brown (eds.), Vol. 1, pp. 31-91, MIT Press, Cambridge, MA, 1979.
10. J. DeKleer and J. S. Brown, Foundations of Envisioning. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 434–437, AAAI, Menlo Park, CA, 1982.
11. A. L. Brown, Qualitative Knowledge, Causal Reasoning, and the Localization of Failures, MIT AI Lab AI-TR-362, Ph.D. thesis, November 1976.
12. D. McDermott and R. Brooks, ARBY: Diagnosis with Shallow Causal Models. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 370–372, AAAI, Menlo Park, CA, 1982.
13. R. R. Cantone, F. J. Pipitone, W. B. Lander, and M. P. Marrone, Model-Based Probabilistic Reasoning for Electronics Troubleshooting. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pp. 207–211, AAAI, Menlo Park, CA, 1983.
14. W. R. Simpson and H. S. Balaban, The ARINC Research System Testability and Maintenance Program (STAMP). In *Proceedings of the 1982 IEEE Autotestcon Conference*, IEEE, New York, 1982.
15. J. J. King, Artificial Intelligence Techniques for Device Troubleshooting. Computer-Science Laboratory Technical Note Series, CSL-82-9 (CRC-TR-82-004), Hewlett Packard, Palo Alto, CA, 1982.
16. E. H. Shortliffe, *Computer-Based Medical Consultations: MYCIN*, Elsevier, New York, 1976.
17. R. O. Duda, P. E. Hart, K. Konolige, and R. Reboh, A Computer-Based Consultant for Mineral Exploration. Artificial Intelligence Center, SRI International, Menlo Park, CA, 1979.
18. G. Shafer, *A Mathematical Theory of Evidence*, Princeton University Press, Princeton, NJ, 1976.