01 Dec 1988

A System for the Diagnosis of Faults using a First Principles Approach

Barbara A. Smith

Ralph W. Wilkerson
Missouri University of Science and Technology, ralphw@mst.edu

A SYSTEM FOR THE DIAGNOSIS OF FAULTS USING A FIRST PRINCIPLES APPROACH

B. A. Smith* and R. W. Wilkerson

CSc-88-10

Department of Computer Science

University of Missouri-Rolla

Rolla, Missouri 65401 (314)341-4491

*This report is substantially the Ph.D. dissertation of the first author, completed December 1988.

ABSTRACT

One of the primary areas of application of Artificial Intelligence is diagnosis. Diagnosis from first principles is a diagnostic technique which uses knowledge of the designed structure and function of a device to determine the possible causes of the malfunction.

This work builds on the foundation of a theory of diagnosis by implementing and extending the theory. A correction to the algorithm which defines the theory is presented. The theory is extended for multiple sets of observations of the system and measurement data.

A fundamental problem in diagnosis is selecting the measurement which will be of the most benefit in reducing the number of competing diagnoses for a system. A heuristic which selects a component whose measurement is likely to be beneficial in isolating the actual diagnosis is also presented.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

I. INTRODUCTION

The computer has become an important tool used in the design and manufacture of complex devices. When these devices fail, however, their complexity makes it difficult to diagnose the problem and determine the cause. The complex design information in a machine usable form can serve as the foundation of a system for diagnosis.

There are three parts to the diagnostic problem. The first is to determine whether a system is exhibiting the correct behavior in a given situation. The second aspect of diagnosis is to determine what could be causing the observed misbehavior. The third is that of diagnostic testing. When there is more than one potential diagnosis, it is the goal of diagnostic testing to establish tests which will confirm or eliminate some of the multiple diagnoses. The test designer may suggest measurements to be taken under current conditions or observations to be made under different conditions. For example, the output(s) of a circuit may be observed under different sets of inputs. An automated diagnostician must address each of these issues. However, the area of diagnostic testing has not been explored to the same extent as the first two areas.

A change may be on the horizon for automated diagnostic reasoning. The first, and to this time most successful, attempts to replicate human performance in the area of diagnostic reasoning have involved using human experience and knowledge of the problem domain. The knowledge and experience of human experts in a particular domain are captured in the form of rules and implemented as a rule-based expert system. While rule-based expert systems have achieved significant results in a number of domains and are widely used, such systems have a number of weaknesses. In particular, if a rule-based expert system is to be developed, a human expert for the domain must be available and his or her knowledge extracted and coded as "rules" by

a knowledge engineer. This is a difficult process since an expert is not always conscious of the rules he or she is applying in a given situation.

A different approach to diagnosis has been termed reasoning from first principles or reasoning from deep knowledge. In this approach, the automated diagnostician uses a description of the system structure and observations describing the system's performance to determine if any faults are apparent. If there is evidence that the system is faulty, the diagnostician uses the system description and observations to ascertain which component(s), if faulty, would explain the behavior. The first principles approach clearly addresses these first two aspects of the diagnostic process. However, there are few results which address diagnostic testing within this framework.

Diagnosis from first principles is a more recent approach to diagnostic reasoning than the more well-known expert system approach. There are many expert systems in use and much is known about when to apply and how to construct expert systems. However, issues such as these need to be addressed within the context of diagnosis from first principles.

Within the artificial intelligence community, a dichotomy seems to exist between those supporting the use of diagnosis from first principles and those supporting the use of so-called shallow reasoning systems. However, it may be that a hybrid approach to diagnosis is ultimately the most successful. The strengths of the one method can cover some of the weaknesses of the other.

Human experts in some domains use both methods. It is well known that a technician diagnosing a malfunctioning device first applies heuristics and experiential knowledge (a rule-based approach). If the rule-based approach fails, the technician will turn to schematics and a more precise model and reason from that model and the observations (a first principles approach) to determine the diagnosis. Work related to

the development of automated diagnosticians which employ a dual approach have recently begun to appear in the literature ([*Ha*88], [*KH*88], [*LL*88]). However, the level of development of first principles methods must be improved in order to build powerful hybrid systems.

What is now termed reasoning from first principles first appeared in the work of de Kleer [*dK*76] on the diagnosis of faults in circuits. The primary application area of diagnosis from first principles has been manufactured devices and especially circuits. Major contributions to the development of this type of diagnostic reasoning have been made by Davis [*Da*84], de Kleer and Williams [*DW*87], Genesereth [*Ge*84], and Reiter [*Re*87]. The significance of diagnosis from first principles and more general reasoning about physical systems can be seen in the literature. A special issue of the Journal of Artificial Intelligence [*AI*84] was devoted to reasoning about physical systems. Also, the topic of a special issue of IEEE Transactions on Systems, Man, and Cybernetics [*SM*87] is causal and diagnostic reasoning.

The purpose of this work is to build on the theory of diagnosis from first principles which was developed by Reiter. Chapter II reviews the previous work related to this type of diagnosis.

First principles diagnosticians reason from a description of the designed structure and function of a device to determine diagnoses. Thus, the crucial components of the diagnostician are (1) the representation used to describe structure and function and (2) the reasoning component. A representation based on predicate calculus is discussed in Chapter II. Various types of inference mechanisms are presented in Chapter III.

Chapter IV contains an analysis of Reiter's approach to diagnosis and a correction of an error in his work. Reiter did not implement a diagnostician based on

his theory. Some results of the implementation developed within the context of this work are also presented.

Three extensions to the theory of diagnosis from first principles are discussed in Chapter V. The extensions are: ways of handling multiple sets of observations, a method for determining the effect of new measurement information on diagnoses, and a heuristic for selecting where to take a measurement in a device.

Chapter VI contains implementation notes and several example circuits which have been diagnosed using the system. Chapter VII concludes the work and suggests directions for future research.

II. DIAGNOSIS USING A FIRST PRINCIPLES APPROACH

A. INTRODUCTION

The hallmark of diagnosis from first principles is the use of information describing the structure of the device, its component parts and interconnections, and its intended behavior. The fact that the system description consists exclusively of information about correct behavior is significant. Thus, it is not necessary to know the ways in which a device or component might fail in order to use the the approach of diagnosis from first principles. However, if the types of failure can be enumerated, that information can be included in the model and used by the diagnostician. The fact that the possible faults need not be known means that, unlike rule-based systems, a first principles diagnostician can deal with faults which have never manifested themselves before.

Diagnosis from first principles is not applicable to all domains. Not all domains have a finite set of principles which define the domain, or the principles may not be known. However, where applicable, the first principles approach can simplify the process of building diagnosticians for new devices, thus allowing an automated diagnostician to be available sooner than would be the case for a rule-based system. The development of a rule-based system requires lead time while human expertise is gained. In the structure-based approach, however, once a reasoning component for the general domain is developed, all that is needed is the specification of the design information. For manufactured devices, the design information is usually available.

There are several advantages of the first principles approach in addition to those already mentioned. A detailed discussion of the some of the advantages is presented by Davis [*Da*84]. First, the diagnostic technique offers a high degree of device independence as compared to the device specific nature of rule-based systems.

Second, the requirements for a structure-based approach are clear. Design information describing the structure and behavior of the device must be completely defined. It is easier to ascertain when these requirements have been met than to determine whether all of the essential rules have been provided by a human expert when building a rule-based system. Further, since a rule-based diagnostic system is symptom driven, it may not be possible to define all of the symptoms and rules.

It is also much easier to define the limits of a diagnostician based on first principles. The rules of an expert system can interact in non-obvious ways, so the limitations of the system are not always apparent. The interaction of the rules also make it difficult to maintain the rule base when a device changes. Such changes can be readily handled in a system based on first principles by simply updating the design information.

The advantages of the first principles approach do have a cost, however. The approach is not applicable to all diagnostic domains. Even where applicable, the computational cost is high. All of the implemented diagnosticians use simplifying assumptions to reduce the complexity of the diagnostic process. As a result, the completeness of the diagnostician cannot be guaranteed. Even if these assumptions were not applied, the currently available reasoning mechanisms are not powerful enough to guarantee completeness.

When design information is used in diagnosis, it is assumed that the design has been verified. The diagnostic process is not intended to uncover design flaws, but certainly the general scrutiny of the process may lead to the discovery of such flaws by the human diagnostician. Current systems operate under the assumption that the device is manufactured correctly and that the physical structure matches the design description. This limits the types of faults which can be diagnosed. In general, automated diagnosticians reasoning from design information cannot identify problems

such as bridge faults where there are connections present in the device that are not represented in the design description. Some progress has been made in developing a first principles diagnostician which can identify faults such as bridge faults and shorts. See [*Da*84] for a discussion of this.

As will be seen, the system description of a device to be diagnosed is quite detailed and the description of even simple devices is very large. However, there are two aspects of the design process which can lessen these problems.

First, complex devices are often designed in a hierarchical manner. The general, overall structure of the device is developed first, illustrating the interconnections of large components. The design of these components is then treated as a separate design problem. Diagnosis can be handled in the same hierarchical fashion. The fault is first isolated to a component at a high level in the hierarchy. The diagnosis of the fault(s) within that component becomes a different diagnostic problem. Genesereth [*Ge*84] discusses diagnosis as a hierarchical process. It should be noted that the complexity of the diagnostic problem is not uniform across all levels of the hierarchy. High level components generally have complex functions and behavior which make it more difficult to isolate the fault. Nonetheless, the extremely large number of subcomponents at the lower levels in the hierarchy make the hierarchical approach not only effective but necessary.

Secondly, many complex devices, particularly electronic components, are designed through the use of computer-aided design tools. The results of interactive design sessions can form the basis of the design description information required by the automated diagnostician. Diagnosis is not tied to any single description format. Any description which captures the structure and operation of the device and can be used by the reasoning component of the diagnostician is acceptable. For example, when diagnosing computer hardware, the design information required by the

diagnostician could be extracted from the hardware description language data which describes the device.

Because of the amount of design information necessary for diagnosis from first principles, the ultimate success of this diagnostic approach will depend upon capturing and making use of already stored design information. The development and acceptance of a standard hardware description language for electronic components will aid in this. The specification of the design information is too complex and tedious to be done by humans working from charts and diagrams. Thus, the process of specifying the design to the diagnostician must be automated.

B. REVIEW OF PREVIOUS WORK ON DIAGNOSIS FROM FIRST PRINCIPLES

In the following sections, diagnosis from first principles as developed by Genesereth, Davis, de Kleer and Williams, and Reiter is presented. The approach of each of these researchers is first discussed individually and then comparisons are made. Genesereth's work is described in more detail than the work of Davis and de Kleer and Williams. Much of what is described in conjunction with Genesereth's work is common to the work of the others and will be referred to in the summary of their work.

One device which appears throughout the literature to illustrate diagnosis from first principles is a full adder. This simple circuit is shown in Figure 1 and will be used for examples in this work. Several other circuits are included in Chapter VI and these will also be used for examples.

In the circuit examples, a naming convention is employed. The name of a component is one or more letters which describe the component type, for example EX for EXCLUSIVE-OR, and a distinguishing number. Components of each type are

numbered as they occur in the device. The inputs and outputs of a component are named IN1 through IN*m* and OUT1 through OUT*n* according to their relative position from top to bottom.

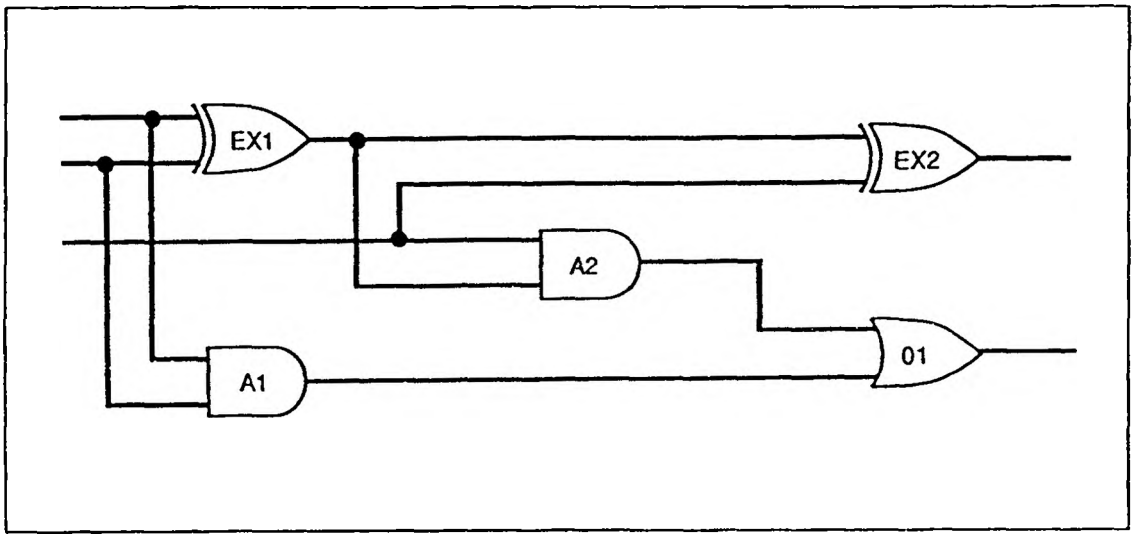

Figure 1. Diagram of a full adder

1. Diagnostic Method of Genesereth.

The major reference for Genesereth's work is [*Ge*84] which is an expanded version of [*Ge*82]. Genesereth has developed an automated diagnostic program called DART. A "tester" who can observe and possibly manipulate the device to be diagnosed provides the diagnostic session information to DART. One of the significant contributions resulting from the development of DART is the use of a hierarchical approach in the diagnostic process. Another contribution is in the area of diagnostic testing. Genesereth proposes a method for suggesting tests, whose outcome may result in a decrease in the number of possible diagnoses.

The design description language and notation which is used by the DART program is based on prefix predicate calculus. The description (structural and behavioral) of the full adder which DART uses is shown in Figures 2 through 4. The full adder is referred to as the device F1 in the description. Variables are indicated by lower case letters. The lowest level components of the device are the EXCLUSIVE-OR gates EX1 and EX2, the AND gates A1 and A2, the OR gate O1, and the connections between these components. The fact that all connections must be explicitly defined affects the type of faults which can be diagnosed. Note that the description is quite lengthy and is clearly based on first order logic.

```
(EXORG EX1)
(EXORG EX2)
(ANDG A1)
(ANDG A2)
(ORG O1)
(CONN (IN 1 F1) (IN 1 EX1))
(CONN (IN 1 F1) (IN 1 A1))
(CONN (IN 2 F1) (IN 2 EX1))
(CONN (IN 2 F1) (IN 2 A1))
(CONN (IN 3 F1) (IN 2 EX2))
(CONN (IN 3 F1) (IN 1 A2))
(CONN (OUT 1 EX1) (IN 1 EX2))
(CONN (OUT 1 EX1) (IN 2 A2))
(CONN (OUT 1 A1) (IN 2 O1))
(CONN (OUT 1 A2) (IN 1 O1))
(CONN (OUT 1 EX2) (OUT 1 F1))
(CONN (OUT 1 O1) (OUT 2 F1))
```

Figure 2. Structural description of the full adder components and connections

```
(IF  (AND (VAL (IN 1 F1) t OFF) (VAL (IN 2 F1) t OFF)
          (VAL (IN 3 F1) t OFF))
     (AND (VAL (OUT 1 F1) t OFF) (VAL (OUT 2 F1) t OFF)))

(IF  (AND (VAL (IN 1 F1) t OFF) (VAL (IN 2 F1) t OFF)
          (VAL (IN 3 F1) t ON))
     (AND (VAL (OUT 1 F1) t ON) (VAL (OUT 2 F1) t OFF)))

(IF  (AND (VAL (IN 1 F1) t OFF) (VAL (IN 2 F1) t ON)
          (VAL (IN 3 F1) t OFF))
     (AND (VAL (OUT 1 F1) t ON) (VAL (OUT 2 F1) t OFF)))

(IF  (AND (VAL (IN 1 F1) t OFF) (VAL (IN 2 F1) t ON)
          (VAL (IN 3 F1) t ON))
     (AND (VAL (OUT 1 F1) t OFF) (VAL (OUT 2 F1) t ON)))

(IF  (AND (VAL (IN 1 F1) t ON) (VAL (IN 2 F1) t OFF)
          (VAL (IN 3 F1) t OFF))
     (AND (VAL (OUT 1 F1) t ON) (VAL (OUT 2 F1) t OFF)))

(IF  (AND (VAL (IN 1 F1) t ON) (VAL (IN 2 F1) t OFF)
          (VAL (IN 3 F1) t ON))
     (AND (VAL (OUT 1 F1) t OFF) (VAL (OUT 2 F1) t ON)))

(IF  (AND (VAL (IN 1 F1) t ON) (VAL (IN 2 F1) t ON)
          (VAL (IN 3 F1) t OFF))
     (AND (VAL (OUT 1 F1) t OFF) (VAL (OUT 2 F1) t ON)))

(IF  (AND (VAL (IN 1 F1) t ON) (VAL (IN 2 F1) t ON)
          (VAL (IN 3 F1) t ON))
     (AND (VAL (OUT 1 F1) t ON) (VAL (OUT 2 F1) t ON)))
```

Figure 3. Behavioral description of the full adder

The last proposition of Figure 4 defines the connections to be ideal. Thus the program cannot diagnose faulty behavior resulting from non-ideal connections, for example, shorts. Theoretically, the DART program could reason with the possibility that connections other than those explicitly defined existed. However, this would necessitate the use of non-monotonic reasoning and result in a much larger number

```
(IF  (AND (ANDG d) (VAL (IN 1 d) t ON) (VAL (IN 2 d) t ON))
     (VAL (OUT 1 d) t ON))
(IF  (AND (ANDG d) (VAL (IN 1 d) t OFF))
     (VAL (OUT 1 d) t OFF))
(IF  (AND (ANDG d) (VAL (IN 2 d) t OFF))
     (VAL (OUT 1 d) t OFF))

(IF  (AND (ORG d) (VAL (IN 1 d) t OFF) (VAL (IN 2 d) t OFF))
     (VAL (OUT 1 d) t OFF))
(IF  (AND (ANDG d) (VAL (IN 1 d) t ON))
     (VAL (OUT 1 d) t ON))
(IF  (AND (ANDG d) (VAL (IN 2 d) t ON))
     (VAL (OUT 1 d) t ON))

(IF  (AND (EXORG d) (VAL (IN 1 d) t ON) (VAL (IN 2 d) t ON))
     (VAL (OUT 1 d) t OFF))
(IF  (AND (EXORG d) (VAL (IN 1 d) t ON) (VAL (IN 2 d) t OFF))
     (VAL (OUT 1 d) t ON))
(IF  (AND (EXORG d) (VAL (IN 1 d) t OFF) (VAL (IN 2 d) t ON))
     (VAL (OUT 1 d) t ON))
(IF  (AND (EXORG d) (VAL (IN 1 d) t OFF) (VAL (IN 2 d) t OFF))
     (VAL (OUT 1 d) t OFF))

(IF  (AND (VAL (IN 1 d) t x) (VAL (IN 2 d) t y) (VAL (OUT 1 d) t z)
          (VAL (IN 1 d) s x) (VAL (IN 2 d) s y))
     (VAL (OUT 1 d) s z))
(IF  (AND (CONN x y) (VAL x t z))
     (VAL y t z))
```

Figure 4. Behavioral description of gates and connections

of possible diagnoses for the observed system behavior. Abstractions such as the assumption of ideal connections are necessary to keep the diagnostic problem tractable. However, as in any application where details are suppressed, there is the risk that the model does not truly reflect the structure and operation of the device and as a result the true diagnosis is not determined by the diagnostician.

Another assumption which is applied by DART is the non-intermittency assumption. This assumption appears as the next to last proposition in Figure 4. The non-intermittency assumption requires that all components behave consistently when presented the same conditions at different times.

Genesereth classifies the information present in the design into four types. Theoretical information describes the components, connections, and functional behavior of components. Achievable data are conditions which the tester can control, for example, setting inputs to particular values. Observable data correspond to information which can be obtained by the tester but not controlled. An example of this type of data is the observed output(s) of a circuit. The fourth type of information is the collection of simplifying assumptions to be used in the diagnostic process.

As discussed, these assumptions are necessary to keep the diagnostic problem to a reasonable size. However, it may be that advances in the theory of diagnosis and the development of more powerful automated reasoning systems will lessen the need for these assumptions. This is the case with the single fault assumption. Early first principles diagnostic systems ([*GE*84], [*Da*84]) depended heavily on this assumption. More recently developed automated diagnosticians ([*Re*87], [*DW*87]) can operate effectively without it. The assumptions which are commonly applied in a diagnostic setting are the single fault assumption, the non-intermittency assumption, and the assumption that certain parts of the device are not faulted. The proposition which defines the connections as ideal is an example of this last assumption.

The single fault assumption states that if a component is faulted, then only that component is faulted and all other components are functioning correctly. Genesereth provides formal characterizations of each of the simplifying assumptions which the user can include in the system description as appropriate for the domain of the

diagnostic problem. However, the situation is more complex than Genesereth leads his readers to believe.

It is not simply a matter of modifying the device description by adding or deleting a few statements. Whether or not an assumption is included can determine the type of reasoning of which the diagnostician must be capable. While the assumptions do represent efficiency concerns, as Genesereth characterizes them, they also determine how powerful the reasoning component of the diagnostician must be as well as the computational complexity of the process.

The diagnostic operation of DART is based on the determination and explanation of symptoms. A *symptom* is observable data which is inconsistent with the system design and achievable data. A symptom might be, for example, observing an output value to be 1 when the value of the input(s) and the proper operation of the device predict it to be 0. Based on one or more symptoms, DART computes a suspect set which will contain the faulty component(s).

Given a suspect set, DART suggests tests to be carried out to distinguish among the suspects. Genesereth ([*Ge*84], page 422) defines a test to consist of "zero or more propositions to be achieved and at least one proposition to be observed". Thus a test will supply additional measurements or observed data under a different set of conditions. The difficulty of suggesting tests lies in determining a test which will provide new information and whose outcome depends on the suspected components.

Let $a_1, \ldots, a_m$ be the achievable and *ob* the observable data prescribed by the test. A check which DART carries out to determine whether any new information may be provided by the test is to try to prove each of the following propositions.

(IF (AND $a_1, \ldots, a_m$) *ob*)
(IF (AND $a_1, \ldots, a_m$) (NOT *ob*))

If either can be proved, the test will provide no new information.

The above "novelty check" as Genesereth calls it is useful, but other criteria need to be developed. The area of measurement and testing theory is largely unexplored. Certainly, as Genesereth states, considerations such as the cost of a test should be taken into account. Also needed is a means of estimating the value of a test based on its ability to decrease the number of possible diagnoses. Obviously, a test which is expensive to carry out and does little to prune the suspect set has little merit. A heuristic for suggesting measurements within the framework of Reiter's theory of diagnosis is discussed in Section D of Chapter V.

The DART system uses an inference method called resolution residue to carry out inferences. Reiter's theory of diagnosis establishes that the underlying inference mechanism can take any form so long as it is a sound decision procedure for the domain. A strength of the resolution residue procedure within DART is that its use provides a means of generating suggested tests. However, it is not clear that Genesereth's method for suggesting tests can be extended for use with other types of inference mechanisms.

Resolution residue is a direct proof procedure which begins with a set of facts and produces new clauses from them. Resolution residue is sound and complete but in the general case, of course, is not a decision procedure for consistency. However, Genesereth ([*Ge*84], page 427) points out that "Fortunately, the problems that arise in diagnosing most computer hardware faults are decidable."

Singh [*Si*87] has built on Genesereth's DART system. He has worked on improving test generation methods by abstracting information from various levels in the design hierarchy. His method also applies focusing techniques to the resolution residue procedure. Since resolution residue is a general proof procedure, without focusing it can be very inefficient and generate a large amount of useless information. It should be noted that the problems of being unfocused and generating useless

information are not problems which are unique to the resolution residue procedure. All automated reasoning systems share these problems.

2. Diagnostic Method of Davis.

Davis's [*Da*84] work on diagnostic reasoning from first principles is significant for its use of multiple representations of the device to be diagnosed. As a result, the system has been successful in diagnosing bridge faults. Other diagnosticians which reason from descriptions of structure and function have been unable to identify such faults. When a bridge fault or short is present in a device, the structural and behavioral descriptions no longer match the actual device.

Where other researchers on diagnosis from first principles have developed general diagnosticians, the approach of Davis is highly specific to the diagnosis of faults in digital electronic hardware. The extensions necessary to carry out diagnosis in other domains are discussed, but at the time of the paper these extensions had not been implemented.

The diagnostician is based on the assumption that the faulty behavior is caused by a single component. In addition, all components, even if faulty, are assumed to be functioning in a consistent manner over multiple sets of observations. Thus, the non-intermittency assumption is built into the model.

A significant feature of the work is the use of multiple representations. The "structure" description of the device consists of separate representations for physical organization and functional or behavioral organization. The reasoning about behavior is based on what Davis terms paths of causal interaction which are built into the representation. Another important feature is the general concept of adjacency which the model exploits. There are different kinds of adjacency. Components which are physically adjacent may not be functionally adjacent and vice

versa. For example, two chips can be adjacent on a board without necessarily having a designed connection between them. There can, however, be unintentional paths of causal interaction.

In order to deal with the computational complexity of the diagnostic problem, only single faults can be diagnosed. Obviously, the cost of this limitation is a loss of completeness. Given that the focus of the work is the diagnosis of faults in digital electronic hardware, the single fault assumption is not without foundation. Another feature of this domain is that the device was functioning correctly and has now failed. In this case, the cause of the fault is more likely to be a single fault diagnosis than if the device is newly assembled and may have multiple malfunctioning components. Of course, the single fault assumption will not allow diagnosis of a fault which has cascaded and caused several components to fail.

Davis's work preceded the work of Reiter and de Kleer and Williams on the diagnosis of multiple faults. Davis discusses how to remove the assumption of a single fault although it had not been implemented. The method which he proposes would first consider possible single faults. If no candidate diagnosis is found, faults involving two components are considered, then faults involving three components, and so on. It appears that the methods of Reiter or de Kleer and Williams for handling multiple faults could be used to extend Davis's diagnostician.

The inference mechanism is guided by discrepancy detection. A discrepancy is a symptom of a fault representing a difference between predicted and observed behavior. A technique which Davis terms constraint suspension is also used. In constraint suspension, the rules under which a component normally functions are temporarily ignored in order to determine whether there is some behavior of the component, not necessarily correct behavior, which would explain the behavior of the system as a whole.

One difference between this system and other diagnosticians using the first principles approach is in the use of experiential knowledge. Expert diagnosticians in the domain of electronic components classified various categories of failure as to their likelihood. The system first attempts to generate candidates in the category of a localized failure, which is considered the most likely category. If no candidate is found in that category, the system moves to the category of bridge faults, and so forth.

The advantage of such an ordering scheme is that it keeps the diagnostician on a path of reasoning with clear goals. However, it also has some of the disadvantages of rule-based expert systems: the categories are domain specific and will differ depending on the diagnostic area, there must be human experts for the domain, and the specific categories must be enumerated.

As is common in diagnostic work, the inference mechanism is based on constraint propagation. The constraint propagator combines simulation (outputs of a component are determined from its inputs) and inferences (the inputs of a component are predicted from its output values.).

3. Diagnostic Method of de Kleer and Williams.

One of the major contributions of the work of de Kleer and Williams [*DW*87] is that multiple faults can be diagnosed. Another contribution is a means of effectively decreasing the number of possible diagnoses through the use of probabilities of failure of the components. Their work has a number of characteristics in common with the general theory of diagnosis developed by Reiter.

In the earlier diagnostic work, such as that done by Genesereth and Davis, the single fault assumption played a prominent role. Even if it was not strictly part of the model, as was the case in Genesereth's work, it was necessary to keep the size of

the diagnostic problem reasonable. The general diagnostic engine (GDE) system of de Kleer and Williams can deal effectively with the complexity which is introduced by allowing multiple faults.

As in Genesereth's DART system, GDE is guided by symptoms, with a symptom implying which components might be faulty. These components form what is termed a conflict. The components in a conflict cannot all be functioning correctly. An obvious characteristic of a conflict is that a superset of it is also a conflict. However, the GDE system must find and manipulate minimal conflicts. The discovery of symptoms and minimal conflicts leads to the determination of what de Kleer and Williams call minimal candidates. These are diagnoses in Reiter's terminology. Reiter makes use of the concept of a conflict, which was originally developed by de Kleer [*dK*76]. In his theory of diagnosis, Reiter uses the term conflict set instead of conflict.

The process of candidate generation is an incremental one of combining the information gained from new minimal conflicts with the known minimal candidates. Initially, the device is assumed to be working correctly and the empty set is the first minimal candidate. The candidate space is viewed as a lattice. At the lowest level is the empty candidate. Above that are all possible candidates containing a single component, then all possible candidates containing two components, and so on until at the top there is the single candidate consisting of all of the components. Each candidate on a level is linked to its supersets on the level above it.

The goal of candidate generation and diagnosis is to start at the empty candidate and move up the lattice the minimum distance necessary to explain the behavior of the system. If a new minimal conflict is not explained by an existing candidate, then that candidate is replaced by one or more of its supersets. The supersets consist of the existing candidate with one of the elements of the new

conflict added. The candidate generation process divides the candidate space lattice into two parts. Above the dividing line are the supersets of the minimal candidates. Below the dividing line are the sets of components which do not explain the observations. The sets of components on the dividing line define the minimal candidates or diagnoses.

The above process allows the diagnostician to identify multiple faults while at the same time guaranteeing that the multiple fault diagnoses found are minimal sets of components. Note that if the single fault assumption were to be applied, the task of candidate generation would be much simpler. All that would be necessary is that the intersection of the minimal conflicts be found. The members of the resulting set define the possible single faults.

A disadvantage of the candidate generation procedure is that it requires that minimal conflicts be determined by the inference mechanism. In order to guarantee minimality, the inference mechanism must explore potential conflicts in increasing order of size until an inconsistency in the form of a difference between predicted and observed behavior is found. The inference mechanism of GDE contains a number of refinements which make minimal conflict discovery more efficient. One of these is a refinement which exploits the sparseness of the search space.

The inference mechanism of GDE is based on constraint propagation and the supporting environment of a value is recorded with the value. The supporting environment represents the components which determine the value. Recall that the diagnostic approach of de Kleer and Williams requires minimality for the conflicts. In order to guarantee the minimality of the conflicts, the supporting environment of a value must be minimal.

De Kleer and Williams propose a method for suggesting measurements in order to decrease the number of candidates. The method makes use of the probability of failure of the individual components along with the set of components which caused an incorrect value. GDE uses a one step lookahead procedure to estimate the effect of a measurement on the set of candidates. De Kleer and Williams state: ([*DW*87], page 116)

> The initial probabilities of candidates are computed from the initial probabilities of component failure (obtained from their manufacturer or by observation). We make the assumption that components fail independently. (This approach could be extended to dependent faults except that voluminous data is required.)

The probability-based approach for suggesting measurements and determining diagnoses is well known. Most rule-based expert systems use probability information to select the most promising paths of reasoning in determining the diagnosis, order the possible diagnoses according to likelihood, and suggest measurements. However, the use of probability information has the disadvantage that such information may not be available or, if available, may not be complete. Furthermore, simplifying assumptions, such as the assumption that components fail independently, are generally necessary in order to keep computations reasonable. De Kleer and Williams make use of several such assumptions and estimates of probabilities. A heuristic for suggesting measurements which does not require knowledge of the probability of the failure of a component and a further discussion of measurements is presented in Chapter V.

The diagnostician also uses the probability information to guide the candidate generation process. Recall that the candidate space is viewed as a lattice and the diagnostician explores the lattice starting with the empty candidate. It then moves to candidates containing one or more components as necessary to explain the symptoms. The search of the lattice is done in a best-first manner based on the probabilities of

the candidates. A path is abandoned as soon as the probability of the candidates further up the path falls below some limit.

4. Diagnostic Theory of Reiter.

Reiter [*Re87*] has developed what he terms a theory of diagnosis. The goal of the theory is to establish a firm foundation on which to develop automated diagnosticians. His theory appears to be applicable to many diagnostic domains. The focus of this dissertation is the diagnosis of circuit faults. Thus, examples and extensions to Reiter's work will be in that field.

The first point to note about Reiter's work is that it is theoretical in nature. He makes no statements to indicate that an automated diagnostician based on the theory has been built. Further, de Kleer and Williams [*DW87*] refer to Reiter's theory as unimplemented. Part of the work of this dissertation involves an implementation of a diagnostician based on Reiter's theory. This implementation has facilitated the discovery of an error in one of Reiter's algorithms and has allowed for the development and testing of a heuristic for suggesting measurements.

Because of the generality of Reiter's theory, issues which are the central focus of some of the earlier work on diagnosis from first principles are ignored. One of these issues is that of the representation logic. Since the theory is independent of the representation logic, the underlying theorem prover (inference mechanism) can be implemented in a manner appropriate for the diagnostic domain. In contrast, the diagnostic systems of Genesereth, Davis, and de Kleer and Williams seem to be dependent on a particular type of inference mechanism. However, Reiter has not demonstrated that the approach of his theory is more general than the approaches of these other researchers. The examples and representation which he uses are the same as those of the other researchers.

De Kleer and Williams also claim that their diagnostic approach is independent of the type of inference mechanism used. However, this seems unlikely given that their diagnostic system is dependent on finding and manipulating minimal conflicts. It is possible to guarantee minimality when constraint propagation is used. Such a guarantee would be very difficult to achieve for an inference mechanism based on a general proof procedure such as resolution.

Some of the terminology of the diagnostic problem domain has already been presented. In the following section, this terminology is expanded and rephrased within the context of Reiter's general theory. The definitions are taken from [*Re*87].

The concept of a system which is to be diagnosed is central to the first principles approach. A *system* is a pair (SD, COMPONENTS) where SD is the system description represented as a set of first-order sentences and COMPONENTS is a finite set of constants representing the constituent parts of the system.

This approach to diagnosis uses the description of a correctly functioning set of components and does not assume any particular mode of failure. Thus the concept of a malfunctioning component must be very general. The predicate *AB*(component) is used for this purpose. Consider again the example of the full adder. A complete representation of the circuit which makes use of the *AB*normal predicate is shown in Figure 5. Four types of information are included in the description.

The correct behavior of components as a function of their input(s) is described by items 1 through 3. These functions must also be defined. In the case of the full adder, this is accomplished by the inclusion of function definitions for AND, OR, and EXCLUSIVE-OR. The name and type of each component is specified in items 4 through 8. The interconnections of the components are defined in 9 through 15. In the case of the full adder, all values are binary and this constraint is specified by items

1) $ANDG(x) \wedge \neg AB(x) \supset out(x) = and(in1(x), in2(x))$
2) $EXORG(x) \wedge \neg AB(x) \supset out(x) = exor(in1(x), in2(x))$
3) $ORG(x) \wedge \neg AB(x) \supset out(x) = or(in1(x), in2(x))$

4) ANDG(A1)
5) ANDG(A2)
6) EXORG(EX1)
7) EXORG(EX2)
8) ORG(O1)

9) out(EX1) = in2(A2)
10) out(EX1) = in1(EX2)
11) out(A2) = in1(O1)
12) out(A1) = in2(O1)
13) in1(A2) = in2(EX2)
14) in1(EX1) = in1(A1)
15) in2(EX1) = in2(A1)

16) $in1(x) = 0 \vee in1(x) = 1$
17) $in2(x) = 0 \vee in2(x) = 1$
18) $out(x) = 0 \vee out(x) = 1$

Figure 5. System description with the ABnormal predicate

16 through 18. A constraint on the values of the inputs and outputs may be absent or may be quite general, depending on the device to be diagnosed. For example, values might be constrained to be integer or positive in some application and without any constraint in another application.

The generality of the approach and representation does not preclude the use of domain specific information concerning faults. One of the primary advantages of the first principles approach is that it is not necessary to know the ways in which a component can be faulted. However, Reiter states that if such information is available, it can be included in the system description. The general form of such information is:

$$\text{COMPONENT\_TYPE}(x) \wedge \text{AB}(x) \supset \text{FAULT}_1(x) \vee \ldots \vee \text{FAULT}_n(x)$$

where FAULT_1 through FAULT_n enumerate the possible behaviors of the faulted component.

Also necessary for diagnosis are one or more sets of observations of the system. An *observation* is simply defined to be a finite set of first order sentences. In the full adder example, observations provide information about known input and output values, such as:

IN1(EX1) = 1
IN2(EX1) = 0
IN1(A2) = 1
OUT1(EX2) = 1
OUT1(O1) = 0

Reiter does not address the problem of representing and reasoning about multiple sets of observations. This issue is discussed in Section B of Chapter V.

As discussed earlier, the goal of diagnostic work is to determine the component(s) which if ABnormal would explain the observed behavior. Since the system description and observations have an underlying logical representation, the concept of a diagnosis is tied to logical consistency. Formally, Reiter defines a *diagnosis* for a device with constituent COMPONENTS and a system description SD under a set of observations OBS to be a minimal set $\Delta \subseteq$ COMPONENTS such that

$$\text{SD} \cup \text{OBS} \cup \{\neg\text{AB}(c) \mid c \in \text{COMPONENTS} - \Delta\} \cup \{\text{AB}(c) \mid c \in \Delta\}$$

is consistent. A slightly simpler characterization of a diagnosis for (SD, COMPONENTS, OBS) is a minimal set Δ such that

$$\text{SD} \cup \text{OBS} \cup \{\neg\text{AB}(c) \mid c \in \text{COMPONENTS} - \Delta\}$$

is consistent. For a proof of the equivalence of the two definitions see [*Re87*].

Two major points arise from this definition. First, a diagnosis must be minimal. As will be seen, Reiter has developed an elegant means of identifying the minimal sets of components which form the diagnoses. Secondly, in order to identify a diagnosis,

there must be a consistency test for the logic used in the representation in the domain of the diagnostic problem. This second point presents a serious problem since, in general, there is no decision procedure for determining the consistency of a first order logic formula. Does Reiter's approach have any merit? The answer is yes. There is no decision procedure for the general question of consistency but for certain domains the question of consistency is decidable. This is true, for example, in the area of boolean circuits.

There are a number of similarities between the work of Reiter and that of de Kleer and Williams. For example, what Reiter terms a diagnosis, de Kleer and Williams refer to as a minimal candidate. The difference, however, between their work is not just a matter of nomenclature. Reiter's approach appears more general than that of de Kleer and Williams and provides a formal basis for studying diagnosis from first principles.

In order to determine the diagnoses, Reiter makes use of the concept of a conflict set which was developed by de Kleer [*dK*76] in his early work on diagnosis from first principles. Informally, de Kleer described a conflict set as a collection of components, all of which could not be functioning correctly. Reiter provides a formal characterization of a conflict set. A *conflict set* for (SD, COMPONENTS, OBS) is a set of components $\{c_1, \ldots, c_k\}$ such that SD $\cup$ OBS $\cup \{\neg AB(c_1), \ldots, \neg AB(c_k)\}$ is inconsistent. A conflict set is minimal if and only if no proper subset of it is a conflict set for (SD, COMPONENTS, OBS).

Reiter's procedure for determining diagnoses for (SD, COMPONENTS, OBS) is based on determining what he terms the minimal hitting sets for the collection of conflict sets for (SD, COMPONENTS, OBS). He defines hitting sets and the related minimal hitting sets as follows.

> Let C be a collection of sets. A *hitting set* for C is set $H \subseteq \cup_{S \in C} S$ such that $H \cap S \neq \{\}$ for each $S \in C$. A hitting set for C is minimal if and only if no proper subset of it is a hitting set for C.

The following theorem ties together the concepts of minimal hitting sets, conflict sets and diagnoses and forms the foundation of Reiter's theory of diagnosis ([*Re*87], page 84).

> Theorem: $\Delta \subseteq$ COMPONENTS is a diagnosis for (SD, COMPONENTS, OBS) if and only if Δ is a minimal hitting set for the collection of conflict sets for (SD, COMPONENTS, OBS).

Thus the problem of computing diagnoses becomes one of computing the minimal hitting sets for the conflict sets of (SD, COMPONENTS, OBS). Note that the problem is phrased in terms of conflict sets and not minimal conflict sets as is the case in the work of de Kleer and Williams. Since the conflict set returned by the reasoning component need not be minimal, the reasoning component may be simpler. However, it will be shown that the inference mechanism of an automated diagnostician based on this theory will eventually find all of the minimal conflict sets.

Reiter provides an elegant means of computing hitting sets through the use of a hitting set tree (HS-tree). Minimal hitting sets are determined by a pruned HS-tree.

> Let C be a collection of sets. An *HS-tree*, T, for C is a smallest edge-labeled and node-labeled tree with the following properties:
>
> (1) The root of T is labeled by $\surd$ if C is empty. Otherwise the root is labeled by a set of C.
>
> (2) If n is a node of T, define $H(n)$ to be the set of edge labels on the path in T from the root node to node n. If node n is labeled by $\surd$ then it has no successor nodes in T. If node n is labeled by a set Σ of C then for each $\sigma \in \Sigma$, node n has a successor node n_σ joined to node n by an edge labeled by σ. The label for node n_σ is a set $S \in C$ such that $S \cap H(n_\sigma) = \{\}$ if such a set S exists. Otherwise, $\surd$ is the label for node n_σ.

Reiter states that $H(n)$ for a node n labeled by $\surd$ is a hitting set for C and each minimal hitting set for C is $H(n)$ for some node n for which $\surd$ is the label. In diagnosis, only minimal hitting sets are desired. A pruned HS-tree will be constructed

in such a way that only the minimal hitting sets are in the tree. Thus, for any node n in the pruned HS-tree labeled by $\surd$, $H(n)$ is a minimal hitting set.

Some difficulties arise in applying the HS-tree structure to the process of computing diagnoses. First, in the diagnostic process, the sets in the collection are conflict sets for (SD, COMPONENTS, OBS). These sets are not explicitly known and it will be shown that all of the minimal conflict sets must appear in the collection. Secondly, the determination of a set $S \in C$ to label a node is computationally expensive since S is computed by the reasoning component. Therefore, it is necessary to use a method which incrementally builds and prunes the HS-tree so that only minimal hitting sets are found and the number of invocations of the reasoning component is kept small. The method as stated by Reiter is:

(1) Generate the pruned HS-tree breadth-first, generating nodes at any fixed level in the tree in left-to-right order.

(2) Reusing node labels: If node n is labeled by a set $S \in C$, and if n' is a node such that $H(n') \cap S = \{\}$, then label n' by S. Such a node n' requires no access to C. The label of node n' is underlined to indicate that it is a label determined by reusing an existing label.

(3) Tree pruning:

(i) If node n is labeled by $\surd$ and node n' is such that $H(n) \subseteq H(n')$, then close the node n'. A label is not computed for n' nor are any successor nodes generated. A closed node is denoted by $\times$.

(ii) If node n has been generated and node n' is such that $H(n') = H(n)$ then close node n'.

(iii) If nodes n and n' have been respectively labeled by sets S and S' of C, and if S' is a proper subset of S, then for each $\alpha \in S - S'$ mark as redundant the edge from node n labeled by α. Reiter claims that a redundant edge, together with the subtree beneath it, may be removed from the HS-tree

while preserving the property that the resulting pruned HS-tree will yield all minimal hitting sets for C. A redundant edge in a pruned HS-tree is indicated by cutting it with ")("

Reiter's claim that a tree pruned in this manner will yield all of the minimal hitting sets is not correct. The interaction of the closing rules, (i) and (ii) above, and the removal of redundant edges, (iii) above, can result in the loss of minimal hitting sets and therefore, an incomplete diagnostician. This is discussed in detail in Section B of Chapter IV.

In the context of the diagnostic process, an access to the collection of sets C in the HS-tree algorithm is an invocation of the inference mechanism. When a label for a node n must be computed (that is, the node cannot be closed or relabeled) then the underlying reasoning component must return one of two possible values. If there exists a conflict set S such that $H(n) \cap S = \{\}$, then the reasoning component must return S, otherwise the distinguished value $\surd$ is returned. Thus, when invoked, the reasoning component is passed the set COMPONENTS $-$ $H(n)$ as well as the system description and observations. If SD $\cup$ OBS $\cup$ $\{\neg AB(c) \mid c \in$ COMPONENTS $- H(n)\}$ is consistent, then $\surd$ is returned. Otherwise, a conflict set (not necessarily minimal) is returned.

Testing the consistency of SD $\cup$ OBS $\cup$ $\{\neg AB(c) \mid c \in$ COMPONENTS $- H(n)\}$ corresponds to the concept of constraint suspension which Davis used. Since the components in $H(n)$ are considered abnormal, the usual functional constraints on their behavior are not applied.

Consider the example of the full adder under the set of observations specified on page 25. An HS-tree for the example is shown in Figure 6. Clearly, this is not the only HS-tree which could be constructed. The shape of the tree and the number of

invocations of the inference mechanism which are needed to complete the tree depends on the order in which node labels are computed.

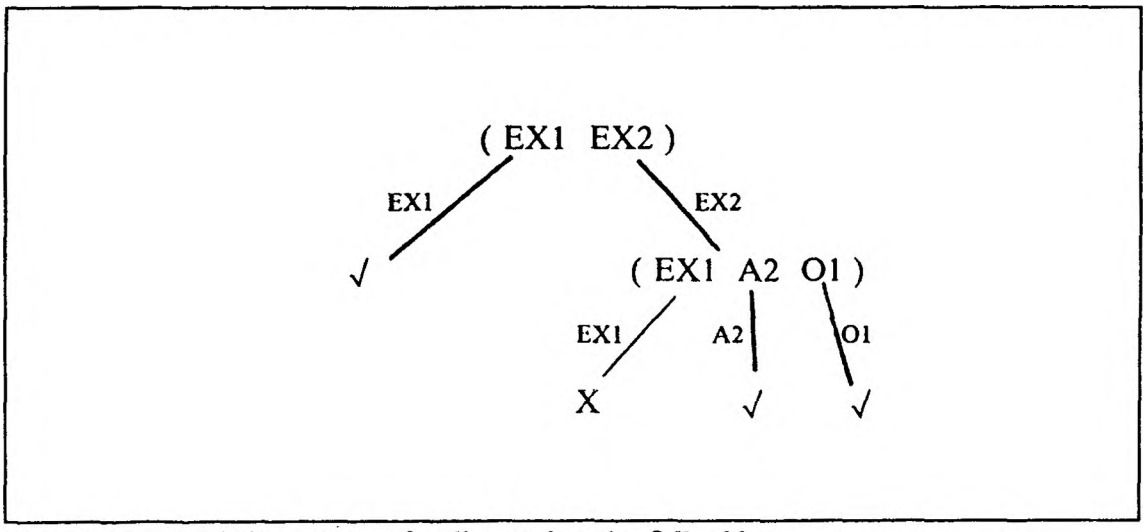

Figure 6. A hitting set tree for diagnosing the full adder

Note that the single fault diagnoses are determined by the nodes labeled with √ at level 1 (the root is level 0) in the tree. If diagnoses of cardinality *k* or less are desired, then the construction of the HS-tree can be halted as soon as level *k* is complete. Thus, the single fault assumption which is prevalent in diagnostic work fits in well with the HS-tree structure.

Another common assumption in diagnostic work is the non-intermittency assumption. The non-intermittency assumption requires components to behave consistently over time. The relationship of the non-intermittency assumption to the HS-tree is discussed in the section on handling multiple sets of observations (Section B of Chapter V).

Recall that the diagnostician developed by de Kleer and Williams makes use of probability information. The candidate lattice is explored in a best-first manner based on the probability of the diagnosis being developed by a particular path. This technique could be incorporated into the construction of the hitting set tree. Rather than building the pruned tree in a breadth first fashion, the node for which the $H(n)$ value represents the partial diagnosis with the highest probability would be expanded first. If the probability of any diagnosis emanating from a particular node falls below some threshold value, that node is not expanded.

When multiple diagnoses are indicated by the observations, it is desirable to reduce the number of diagnoses by taking additional measurements on the system. Genesereth [*Ge*84] presents a method for suggesting measurements. Part of that method involved the used of the novelty check which was discussed in the section on Genesereth's work. The use of the novelty check, however, is directly tied to the use of resolution residue as the inference mechanism. It is not clear whether the novelty check can be extended for an inference mechanism based on some other technique.

Reiter provides a general theorem concerning how new measurement information relates to the diagnoses for a system.

> Theorem: Suppose every diagnosis for (SD, COMPONENTS, OBS) predicts one of Π, $\neg\Pi$. Then:
>
> 1) Every diagnosis for (SD, COMPONENTS, OBS) which predicts Π is a diagnosis for (SD, COMPONENTS, OBS $\cup$ $\{\Pi\}$).
>
> 2) No diagnosis for (SD, COMPONENTS, OBS) which predicts $\neg\Pi$ is a diagnosis for (SD, COMPONENTS, OBS $\cup$ $\{\Pi\}$).
>
> 3) Any diagnosis for (SD, COMPONENTS, OBS $\cup$ $\{\Pi\}$) which is not a diagnosis for (SD, COMPONENTS, OBS) is a strict superset of some diagnosis for (SD, COMPONENTS, OBS) which predicts $\neg\Pi$. In other words, any new diagnosis resulting from the new measurement Π will be a strict superset of some old diagnosis which predicted $\neg\Pi$.

Reiter's theorem on measurements is important to an overall theory of diagnosis, but it does not address a very significant question. That question is: At what points in a device should measurements be taken in order to provide the most useful information in the diagnostic process? A measurement heuristic is discussed in Section D of Chapter V.

The difficulty of reducing the number of diagnoses is illustrated by point three of the theorem. While it is true that measurements can confirm or reject existing diagnoses, the number of diagnoses can increase. Further, the number of components involved in a particular diagnosis can also increase resulting in new diagnoses. Reiter raises an important research question with respect to measurements. Is is possible to identify which measurement(s) will simply filter existing diagnoses and not result in any new diagnoses? The answer to this open question would be a significant result for diagnosis from first principles.

Another question which Reiter raises is how to make use of the existing HS-tree with the new measurement information. Because of the amount of computation necessary to build an HS-tree, it is desirable to make use of already known value information when rejecting existing diagnoses or computing new diagnoses. A method for extending the existing HS-tree to determine the diagnoses in the presence of measurement information is discussed in Section C of Chapter V.

C. REPRESENTATION OF A DEVICE

1. Introduction.

A language which is suitable for representing the type of information found in an annotated schematic must be selected as the representation language for a diagnostician based on first principles. The representation of the device must convey

the information necessary to make simulation of the the device possible and allow for the determination of conflict sets when the device is malfunctioning.

In the life cycle of a device, different types of data are required in different phases. Major tasks in the life cycle include design verification, manufacturing, production line quality control testing, and diagnosis. While there is clearly some overlap among these tasks, they all require a different view of the device. Some of the information necessary to carry out one phase is relevant to other phases but much of the information is irrelevant. Therefore, a single, universal description of a device that includes all of the information necessary for all of the phases is not practical.

Another problem inherent in representing a device symbolically is the amount of information which must be included. This is particularly true of complex devices. However, these are precisely the devices for which automated diagnosticians, especially those utilizing the first principles approach, are most needed. Certainly the use of hierarchical design information can alleviate this problem by reducing the amount of information needed to describe any given level in the hierarchy as compared to the entire device.

Singh [*Si*87] has investigated the use of design information and description languages. He examines the problem of making use of essential information in the descriptions of a device which are used in the various phases of the life cycle. He also considers automatically reformulating design information in order to make better use of it. The goal of his work is the efficient generation of tests to verify that a device is manufactured correctly.

A discussion of the requirements for a description language which is appropriate for diagnosis begins with a device. A device is a set of components with a defined and unchanging physical configuration. Along with a physical structure, the

components of the device and the device itself have defined functions when operating correctly. If the components are rearranged or the function of a component is changed, a new device results.

In any language, a distinction is made between the syntax of a language and its semantics. A design description language should not admit ambiguous sentences, as that would imply that the device does not have a definite physical structure or function. A desirable characteristic of a device description language for diagnosis is device independence. The advantage of using a device independent language is that the various descriptions corresponding to the different hierarchical views can be provided in the same language.

In general, a language for specifying information to be used in manufacturing a device cannot be device independent. Whether it is practical to maintain a device independent language for diagnosis is an open question. In order for the first principles approach to be widely applicable, automated translation of information expressed in a design language to the descriptive language of the diagnostician must be accomplished.

2. Representation Based on Predicate Calculus.

The use of predicate calculus provides a device independent language for describing the device and making inferences about its operation. The syntax and semantics of sentences in the predicate calculus are well-defined. The information expressed by the design includes the specific sentences in the design and all other sentences which can be derived from the design by sound inference rules.

There are different types of symbols: constants, variables, functions, and predicates. Functions and predicates have associated with them an arity. These symbols can be combined to form terms where

1) A constant is a term
2) A variable is a term.
3) If f is an n-ary function and $t_1, \ldots, t_n$ are terms then $f(t_1, \ldots, t_n)$ is a term.
4) Terms are generated by a finite number of applications of rules 1 through 3.

Functions and predicates may be written using infix rather than the prefix notation used above. Predicates are mapped to *TRUE* or *FALSE* and inferences are made based on predicates.

If P is an n-ary predicate and $t_1, \ldots, t_n$ are terms, then $P(t_1, \ldots, t_n)$ is an atom. Atoms are themselves sentences and can also be combined with logical operators to form other sentences. The logical operators used in this work are negation ($\neg$), conjunction ($\wedge$), disjunction ($\vee$), and implication ($\supset$).

The expressive power of the predicate calculus is due to the use of variables and the quantification of those variables. Variables in the context of sentences are universally or existentially quantified. A variable, v, which is universally quantified is denoted by adding $(\forall v)$ to the beginning of the sentence. The meaning of the resulting sentence is that the sentence is true regardless of the object which is used for the variable v. If the variable v is existentially quantified, $(\exists v)$ is added to the beginning of the sentence. The meaning of the resulting sentence is that there is at least one object which can be used for v that makes the sentence true. When describing combinational devices, the use of universal rather than existential quantifiers in the design description is typical. The reader is referred to [*CL*73] or [*LP*81] for a more detailed discussion of predicate calculus.

As with any language, the meaning of a sentence in the predicate calculus is defined through the semantics of the language. The basic element of the semantics is an interpretation. The interpretation of a sentence provides a domain and an assignment to constants, functions, and predicates which occur in the sentence. In the case of Boolean combinational circuits, the interpretation which is applied to the description of the circuit is the standard interpretation of Boolean algebra.

Consider again the description provided by Reiter of the full adder which was presented in Figure 5. The variables occurring on each line of the description are distinct and are universally quantified. The constants which appear in the description are 0, 1, EX1, EX2, A1, A2, and O1. When inferences are carried out on the description, the individual sentences are combined to form a single sentence. The single sentence is the conjunction of the simpler sentences with the appropriate distinction of the variables. For readability, = is written as an infix operator.

In the predicate calculus, a sentence is either satisfiable (consistent) or unsatisfiable (inconsistent). A sentence is satisfiable if and only if there is at least one interpretation which makes the sentence true. It is unsatisfiable if and only if no interpretation exists which makes the sentence true. A sentence is valid if and only if every interpretation makes the sentence true.

According to Reiter's characterization of a diagnosis, a diagnosis is a minimal set of components (which are constants in the description) such that under the assumption that only those components are malfunctioning, an interpretation exists which makes the system description consistent. The difficulty lies in determining the consistency of the system description and observations. In general, the question is undecidable.

A language based on predicate calculus has been used as the representation language in first principles diagnosticians. The description given by Genesereth of the full adder (Figures 2 through 4) is based on the predicate calculus. The syntax is modified slightly for the resolution residue inference mechanism. The predicate VAL is used instead of the operator =.

The primary advantage in using predicate calculus as the description language is its generality and robustness. The difficulty lies in developing inference mechanisms which can efficiently use the design information. Genesereth uses a very general

format for the description of the device and then uses focusing techniques within the inference mechanism. Reiter does not discuss the form of the inference mechanism and ignores efficiency concerns.

The approach taken in this work has been to identify the critical areas in the design description and tailor the representation and the inference mechanism to emphasize these areas. An example of a critical area is the definition of connections. It is well known that automated reasoning systems cannot effectively deal with equality. Because a connection represents an instance where equality reasoning is applied, it can be a source of inefficiency. By handling connections separately from the rest of the description, the inference mechanism can reason more effectively.

The disadvantage of specializing the description and inference mechanism is that adding assumptions like the non-intermittency assumption can be more difficult. Genesereth states that assumptions such as the non-intermittency assumption can be included as deemed appropriate by the user. This is true in theory since the representation and inference mechanism are general. The practicality of this is another matter. The non-intermittency assumption and the assumption of ideal connections (see Figure 4) each contain several variables. This will allow these assumptions to enter into many inferences and can cause a general inference mechanism to become unfocused. In addition, if the assumption of ideal connections is not used, the inference mechanism must be capable of non-monotonic reasoning.

In this work, the assumption of ideal connections is built into the model. An area of future research is to investigate ways of removing this assumption without causing a great loss of efficiency. The non-intermittency assumption can be applied or not. However, when the assumption is applied it is not included as a sentence in the design description. Rather, it is part of the inference mechanism. This allows for

control over how reasoning about the assumption is applied. A more detailed discussion of the implementation appears in Chapter VI.

Another question which is related to the representation of devices is how to represent sequential devices. None of the diagnosticians based on first principles have been designed to handle devices which have feedback. It is not known whether the use of predicate calculus as the representation language is appropriate for these devices.

III. INFERENCE MECHANISM

A. INTRODUCTION

An inference mechanism is a procedure which applies inference rules to a collection of assumptions in order to derive additional information. In a diagnostician which is based on Reiter's theory of diagnosis, it is the role of the inference mechanism to determine whether the system description and observations are consistent when a particular set of components is assumed to be functioning in an (unspecified) abnormal manner. If so, the set of components is a diagnosis for the system. Otherwise, the inference mechanism must identify a set of components, known as a conflict set, for which the assumption that the components were functioning correctly resulted in the inconsistency. The conflict set will allow the correct combination of components to be tested so that the set of diagnoses can be determined. Clearly, the inference mechanism plays a central role in diagnosis.

The inference mechanism is a theorem prover, although the domain of the theorem prover is usually specialized. A theorem prover is sound when the conclusions drawn by it are true when the premises are true. In a theorem proving environment, the goal is to find the proof. If a set of clauses is to be shown to be valid, the clauses are negated and a refutation is sought. If a theorem prover is guaranteed to find a refutation when one exists, the theorem prover is said to be refutationally complete.

The behavior of a theorem prover for first order logic cannot be predicted if a refutation does not exist. The theorem prover may halt. If so, and if the theorem prover is complete, then it can be correctly concluded that the original clauses are satisfiable. However, the theorem prover may never halt. The question of determining the consistency of a set of clauses in first order logic is undecidable; that

is, there is not procedure which can in all cases determine whether the clauses are consistent or inconsistent. Since there are, however, procedures which are sound and complete for determining inconsistency in first order logic, the problem is said to be semi-decidable.

B. THEOREM PROVING IN FIRST ORDER LOGIC

In the theory of diagnosis developed by Reiter, some very general criteria were established for the inference mechanism of an automated diagnostician based on the first principles approach. The inference mechanism must be a decision procedure for testing the consistency of the clauses which describe the system under a set of observations. If an inconsistency is found, the components contributing to that inconsistency must be returned as a conflict set.

In most first principles diagnostic systems currently under development, the representation language is based on first order logic. First order logic provides for a rich representation, but two major problems arise. First, even when in a decidable domain such as boolean circuits, a theorem prover for first order logic generally is not a decision procedure. The second problem is that a general theorem prover is too inefficient for use in a diagnostic setting. In developing his theory, Reiter chose to ignore issues of representation, implementation, and computational costs. However, even very simple diagnostic problems illustrate that these issues should be of major concern. Consider two examples: the full adder shown in Figure 1 and the more complex two bit adder-subtracter shown in Figure 31. These examples will illustrate the need for a specialized reasoning system, rather than a general theorem prover, as the inference mechanism.

1. ITP.

The Interactive Theorem Prover (ITP) [*OL*84] is a general theorem proving system which was developed by the Mathematics and Computer Science Division of Argonne National Laboratory. The system is designed to provide a convenient environment for research in automated theorem proving. It is a clause-based reasoning system which supports several inference techniques and strategies. The ITP input clauses for the example circuits are listed in Figures 7 through 10. The input clauses for the full adder were developed directly from Reiter's system description of the device. Clauses which are input to ITP are divided into four groups: Axioms, Set of Support, Demodulators, and Have Been Given. In both circuit examples, there are no clauses in the Have Been Given group.

Axioms:

System description axioms from Figure 8

Set of Support:

-AB(EX1);
-AB(EX2);
-AB(A1);
-AB(A2);
-AB(O1);
ANDG(A1);
ANDG(A2);
EXORG(EX1);
EXORG(EX2);
ORG(O1);
OUT(EX1) = IN1(EX2);
OUT(EX1) = IN2(A2);
OUT(A2) = IN1(O1);
OUT(A1) = IN2(O1);
IN1(A2) = IN2(EX2);
IN1(EX1) = IN1(A1);
IN2(EX1) = IN2(A1);

Demodulators:

IN1(EX1) = 1;
IN2(EX1) = 0;
IN2(A1) = 0;
IN1(A1) = 1;
IN2(EX2) = 1;
IN1(A2) = 1;
OUT(O1) = 0;
OUT(EX2) = 1;

The function definitions for Boolean algebra from Figure 9 are also included as demodulators.

Figure 7. ITP clauses describing the full adder

Axioms:

```
x = x;
-(1 = 0);
-(0 = 1);
-EXORG(x) | AB(x) | OUT(x) = EXOR(IN1(x),IN2(x));
-ANDG(x) | AB(x) | OUT(x) = AND(IN1(x),IN2(x));
-ORG(x) | AB(x) | OUT(x) = OR(IN1(x),IN2(x));
IN1(x) = 0 | IN1(x) = 1;
IN2(x) = 0 | IN2(x) = 1;
OUT(x) = 0 | OUT(x) = 1;
```

Figure 8. Axioms which form the basis of the system description

Demodulators:

```
AND(0,x) = 0;
AND(x,0) = 0;
AND(1,1) = 1;
OR(1,x) = 1;
OR(x,1) = 1;
OR(0,0) = 0;
EXOR(x,x) = 0;
EXOR(1,0) = 1;
EXOR(0,1) = 1;
```

Figure 9. Definition of Boolean functions included as demodulators

The representation of these circuits is heavily dependent on equality. While humans can handle equality very easily, automated reasoning systems cannot. Paramodulation and demodulation are techniques used by automated reasoning systems for dealing with equality. Paramodulation [*WR*69] is an inference rule which is applied to clauses which contain the equality relation. Demodulation [*Wo*67] is a rewriting process which is designed to simplify clauses through equality substitution. It is through the use of paramodulation and demodulation that the predicate *equals* takes on special significance.

Axioms:

System description axioms from Figure 8

Set of Support:

EXORG(EX1);

EXORG(EX6);
ANDG(A1);

ANDG(A4);
ORG(O1);
ORG(O2);
-AB(EX1);

-AB(O2);
IN1(EX2) = OUT(EX1);
IN1(A1) = OUT(EX1);
IN1(EX3) = OUT(EX2);
IN2(A2) = OUT(EX2);
IN1(O1) = OUT(A2);
IN2(O1) = OUT(A1);
IN1(EX5) = OUT(EX4);
IN1(A3) = OUT(EX4);
IN1(EX6) = OUT(EX5);
IN1(A4) = OUT(EX5);
IN1(O2) = OUT(A4);
IN2(O2) = OUT(A3);
IN2(A4) = OUT(O1);
IN2(EX6) = OUT(O1);

Demodulators:

IN1(EX1) = 0;
IN2(EX1) = 0;
IN1(A2) = 0;
IN2(EX3) = 0;
IN2(EX2) = 1;
IN2(A1) = 1;
OUT(EX3) = 0;
IN1(EX4) = 0;
IN2(EX4) = 1;
IN2(EX5) = 1;
IN2(A3) = 1;
OUT(EX6) = 1;
OUT(O2) = 1;

The function definitions for Boolean algebra from Figure 9 are also included as demodulators.

Figure 10. ITP clauses describing the two bit adder-subtracter

Equality is defined by the axioms of reflexivity, symmetry, transitivity, and substitution. If reasoning about equality is accomplished through the inclusion of these axioms, many inferences are needed to derive clauses which are obvious consequences of the clauses and the properties of equality. In addition, many clauses which are sound, but useless in finding an inconsistency or determining consistency, can be derived from these axioms. Thus, the clause space becomes cluttered and the theorem prover unfocused.

Paramodulation eliminates the need to include the equality axioms except that of reflexivity. (The clause $x = x$ is included as an axiom.) Since the clause space is smaller and the reasoning about equality more natural, the theorem prover is more focused. Demodulation, as a rewriting in place, helps to keep the clauses simpler. However, the problem of reasoning about equality is far from solved. Paramodulation and demodulation are important tools, but developing more powerful reasoning systems which handle equality better is an important research topic. This and other related research questions in the area of automated reasoning are discussed in [*Wo*88].

The inference rule of binary resolution [*Ro*65] is the foundation of clause based theorem proving systems. Binary resolution results in a very small sized inference since only two clauses are used in an inference step. The use of *Unit-Resulting resolution* (UR-resolution) can increase the step size of inferences. UR-resolution combines several clauses simultaneously to produce a unit clause, which is a clause that contains only one literal. In addition, UR-resolution allows only one non-unit clause to take part in the deduction.

The emphasis that UR-resolution places on unit clauses is appropriate for the clauses in the system description of a circuit. The unit equality clauses that are derived will become demodulators which can be used to simplify the clause space. In

addition, many of the initial clauses are unit clauses. It should be noted that UR-resolution is no more powerful than binary resolution in terms of what can be deduced. Anything which can be deduced through UR-resolution can be derived through one or more applications of binary resolution. However, the intermediate clauses which binary resolution creates and retains in the clause space are not present if UR-resolution is used.

The set of support strategy and subsumption can aid in the discovery of the proof. When a set of support is utilized, any clause which is generated by the automated theorem prover must have at least one parent which has support. A clause is said to have support if it is a member of the set of support. As new clauses are generated, they are added to the set of support. If the initial members of the set of support are judiciously chosen, the theorem prover can be kept more focused.

Subsumption allows the theorem prover to delete a clause which is less general than some other clause. This reduces the size of the clause space and retains the simplest clauses. Care must be taken when using subsumption and set of support as a loss of completeness can occur. See [*WO*84] for a discussion of this.

Table I contains statistics for the determination of conflict sets for the two example circuits by ITP. In these cases, the input clauses were known to be inconsistent and thus yield a conflict set. UR-resolution, paramodulation, demodulation, subsumption, and the set of support strategy were used. The CPU times are inference times and do not include the time necessary to set the options, read in the clauses, or display the proof. The conflict sets represent node labels in a hitting set tree and the components listed as abnormal are the collection of components on the path from the root to the node labeled by the conflict set. Note that the conflict sets returned by ITP, or any other general theorem prover, are not

necessarily minimal. The effect of non-minimal conflict sets on the computational cost of the diagnostic problem is discussed in Chapter IV.

Table I. STATISTICS ON THE DETERMINATION OF CONFLICT SETS BY ITP

Circuit	Conflict Set	Input Clauses	New Clauses	Time (sec)	Abnormal Components
Full adder	(EX1 EX2)	43	27	11.36	--
	(A1 A2 O1 EX1)	43	30	11.70	EX2
	(A2 O1 EX1)	43	28	10.70	EX2 A1
Adder-subtracter	(EX1 EX2 EX3)	69	28	13.08	--
	(EX1 EX4 EX5 EX6 A1 A2 O1)	69	43	20.55	EX2
	(EX1 EX5 EX6 A1 A2 A3 A4 O1 O2)	69	250	346.33	EX2 EX4

Even if the performance figures from Table I were acceptable, a general theorem prover is not a decision procedure for determining the consistency of first order logic clauses. When the clause set is satisfiable, the only way that a general theorem prover could determine satisfiabilty is by creating new clauses until no more new clauses result. Clearly, even when this is theoretically possible, it is not practically computable. This work with ITP was done to determine CPU time benchmarks for the determination of conflict sets.

2. Inference Mechanism Based on Term Rewriting.

Hsiang [*Hs*85*a*] has developed a method for refutational theorem proving which is based on term rewriting. His results indicate that in certain domains, such a theorem proving system is very efficient. Further, Hsiang's work has been used by Chandrasekhar et al. [*CP*87] as the basis of a procedure for verifying the design of combinational circuits. The procedure uses theorem proving and makes exhaustive simulation of the circuit is unnecessary. Since the problems of verification and diagnosis are related, it was thought that an inference mechanism based on term rewriting should be investigated. A Common Lisp implementation of a theorem prover based on term rewriting was developed in order to determine its applicability as an inference mechanism in a diagnostician.

A *term rewriting system* is a finite set of directed equations of the form $\lambda \rightarrow \rho$, which are called *rewrite rules* or *reductions*. Here, λ and ρ are terms and the equality is directed by some ordering scheme. A term t can be *reduced* by $\lambda \rightarrow \rho$ if there is a subterm s of t and a substitution σ such that $\lambda\sigma = s$. The substitution is a set of variable-term pairs. When the substitution σ is applied to a term t, for every variable-term pair in σ, whenever that variable occurs in t it is replaced by the associated term. When the term t is reduced by the reduction $\lambda \rightarrow \rho$, the substitution which makes $\lambda\sigma = s$ is applied only to λ. Thus, there is said to be a *match* between λ and s and s is said to be an instance of λ. The reduction is made by replacing s by $\rho\sigma$. A term is *irreducible* or in *normal form* if no rule can be applied to reduce it.

Of particular interest are rewriting systems which possess the properties of finite termination and unique termination. Finite termination means that for all terms, after a finite number of applications of the rewrite rules, an irreducible term is produced. Unique termination means that if a term t can be reduced in two ways to terms r and s, then the normal forms of r and s are the same. A term rewriting

system which satisfies these properties is called a canonical rewriting system or a complete set of reductions.

A complete set of reductions for Boolean algebra using the operations of AND, OR, and NOT does not exist since there is no unique representation of a term under these operations. However, Hsiang showed that by representing Boolean algebra formulas using the Boolean operations of AND ($\wedge$) and EXCLUSIVE-OR ($\oplus$) that a complete set of reductions for Boolean algebra exists using these operations. As a consequence of this representation, every formula in the propositional calculus has a unique normal form. The normal form is either 0, 1, or $t_1 \oplus \ldots \oplus t_n$, where the t_i are products of distinct positive literals. The rewrite rules in the complete set of reductions are:

$$x \oplus 0 \rightarrow x$$
$$x \oplus x \rightarrow 0$$
$$x \wedge 1 \rightarrow x$$
$$x \wedge x \rightarrow x$$
$$x \wedge 0 \rightarrow 0$$
$$x \wedge (y \oplus z) \rightarrow x \wedge y \oplus x \wedge z$$

It should be noted that the operators $\wedge$ and $\oplus$ are both associative and commutative with identity 1 and 0 respectively. This complete set of reductions requires the use of a unification algorithm which incorporates into the unification process the associative and commutative properties of the operators $\wedge$ and $\oplus$. The procedure for determining a complete set of reductions for an equational theory was first developed by Knuth and Bendix [*KB*70]. The Knuth-Bendix completion procedure could not handle equational theories which contained commutative operators. Associative operators could be handled to a limited extent by including a rewrite rule of the form $f(f(x, y), z) \rightarrow f(x, f(y, z))$ for each associative operator f. Note that this does not represent a complete definition of associativity. Peterson and Stickel [*PS*81] extended the Knuth-Bendix completion procedure to handle

associative and commutative theories. Hsiang used the Peterson-Stickel completion procedure to determine the complete set of reductions for Boolean algebra.

Terms involving the Boolean operators implication ($\supset$), disjunction, ($\vee$), negation ($\neg$), and bi-implication ($\Leftrightarrow$) are transformed into terms using the operators $\wedge$ and $\oplus$ by the following rules. The operator $\wedge$ is indicated by the juxtaposition of the operands.

$$x \supset y \rightarrow xy \oplus x \oplus 1$$
$$x \vee y \rightarrow xy \oplus x \oplus y$$
$$\neg x \rightarrow x \oplus 1$$
$$x \Leftrightarrow y \rightarrow x \oplus y \oplus 1$$

The direct application of these rules to transform Boolean terms into their EXCLUSIVE-OR and AND equivalents, however, is very inefficient. Hsiang gives a strategy for carrying out this transformation more efficiently.

The procedure for showing that a sentence Φ is valid is to first find the clausal form of the sentence's skolemized negation. Each clause C_i of $C_1 \wedge \ldots \wedge C_n$ becomes a rewrite rule $C_i \rightarrow 1$. Each C_i is transformed into its equivalent in terms of $\oplus$ and $\wedge$ and reduced to normal form according to the reductions in the complete set of reductions for Boolean algebra. The *N-strategy* refutational procedure is then applied. The N-strategy is similar to the Knuth-Bendix completion procedure. The rules formed from $C_1 \wedge \ldots \wedge C_n$ are overlapped to form new rules until the contradiction $1 \rightarrow 0$ is found. At all times in the process, the rules are kept in normal form and all rules in the set are kept inter-reduced.

The theorem prover is refutationally complete and as such the rule $1 \rightarrow 0$ will result if the input clauses are inconsistent. If the input clauses are consistent, the strategy will either produce infinitely many rules or complete the superposition process and terminate with a finite set of rules.

As originally developed by Hsiang, the theorem prover had no "built in" understanding of equality. Nonetheless, some impressive results for proving theorems in certain domains were obtained. Hsiang [*Hs85b*] has since incorporated some reasoning about equality which is analagous to paramodulation in a resolution based theorem prover.

For this work, a prototype system incorporating demodulation was written. Hsiang discusses several techniques for improving the efficiency of a theorem prover based on rewriting. These include special orderings and lookup procedures, generating unifiers in a specific order, splitting a long rule into two or more shorter rules, and others. It should be noted that not all of these techniques were incorporated in this implementation. Thus, it might be possible to realize significant improvements in run times by using these techniques.

Demodulation was a natural addition to the N-strategy since as part of the N-strategy rules are kept fully reduced by the reductions in the complete set of reductions for Boolean algebra and the other rules in the set. The input included a list of demodulators as well as the input clauses. The input for the full adder and the two bit adder-subtracter examples is shown in Figures 11 and 12, respectively. Since the circuits are Boolean, two axioms which assert that 0 and 1 are distinct constants were added. The input is in clausal form and each input line is a disjunction of literals. The input also contains a list of components. The diagnostician builds the appropriate clauses asserting that a particular component is to be treated as abnormal, AB(c), or functioning correctly, $\neg$AB(c). Note that the function definitions for the Boolean functions are included as demodulators.

```
(setq components '(A1 A2 O1 EX1 EX2))

(setq origrules '(((- = 1 0))
                  ((- = 0 1))
                  ((- ANDG x) (AB x) (= (OUT x) (AND (IN1 x) (IN2 x))))
                  ((- EXORG x) (AB x) (= (OUT x) (EXOR (IN1 x) (IN2 x))))
                  ((- ORG x) (AB x) (= (OUT x) (OR (IN1 x) (IN2 x))))
                  ((ANDG A1))
                  ((ANDG A2))
                  ((EXORG EX1))
                  ((EXORG EX2))
                  ((ORG O1))))

(setq origDML '(((IN1 EX2) (OUT EX1) nil)
                ((IN2 A2) (OUT EX1) nil)
                ((IN1 O1) (OUT A2) nil)
                ((IN2 O1) (OUT A1) nil)
                ((OUT O1) 0 nil)
                ((OUT EX2) 1 nil)
                ((IN1 EX1) 1 nil)
                ((IN2 EX1) 0 nil)
                ((IN1 A2) 1 nil)
                ((IN1 A1) 1 nil)
                ((IN2 A1) 0 nil)
                ((IN2 EX2) 1 nil)
                ((OR 1 x) 1 nil)
                ((OR x 1) 1 nil)
                ((OR 0 0) 0 nil)
                ((AND 0 x) 0 nil)
                ((AND x 0) 0 nil)
                ((AND 1 1) 1 nil)
                ((EXOR x x) 0 nil)
                ((EXOR 0 1) 1 nil)
                ((EXOR 1 0) 1 nil)))
```

Figure 11. Lisp input to the theorem prover for the full adder

When a new rule is formed, a list of particular ancestors of that rule is associated with it. Only ancestors of the form $\neg AB(c)$ need be carried with the rule. A full recreation of the proof is not necessary since only the conflict set is needed. Note that the last element of every demodulator in the input is the distinguished Lisp

```
(setq components '(EX1 EX2 EX3 EX4 EX5 EX6 A1 A2 A3 A4 O1 O2))

(setq origrules '(((- = 1 0))
                  ((- = 0 1))
                  ((- ANDG x) (AB x) (= (OUT x) (AND (IN1 x) (IN2 x))))
                  ((- EXORG x) (AB x) (= (OUT x) (EXOR (IN1 x) (IN2 x))))
                  ((- ORG x) (AB x) (= (OUT x) (OR (IN1 x) (IN2 x))))
                  ((= (IN1 x) 1) (= (IN1 x) 1))
                  ((= (IN2 x) 1) (= (IN2 x) 1))
                  ((= (OUT x) 1) (= (OUT x) 1))
                  ((EXORG EX1))

                  ((EXORG EX6))
                  ((ANDG A1))

                  ((ANDG A4))
                  ((ORG O1))
                  ((ORG O2))))

(setq origDML '(((IN1 EX1) 0 nil)
                ((IN2 EX1) 0 nil)
                ((IN1 A2) 0 nil)
                ((IN2 EX3) 0 nil)
                ((IN2 EX2) 1 nil)
                ((IN2 A1) 1 nil)
                ((IN1 EX2) (OUT EX1) nil)
                ((IN1 A1) (OUT EX1) nil)
                ((IN1 EX3) (OUT EX2) nil)
                ((IN2 A2) (OUT EX2) nil)
                ((IN1 O1) (OUT A2) nil)
                ((IN2 O1) (OUT A1) nil)
                ((OUT EX3) 0 nil)
                ((IN1 EX4) 0 nil)
                ((IN2 EX4) 1 nil)
                ((IN2 EX5) 1 nil)
                ((IN2 A3) 1 nil)
                ((IN1 EX5) (OUT EX4) nil)
                ((IN1 A3) (OUT EX4) nil)
                ((IN1 EX6) (OUT EX5) nil)
                ((IN1 A4) (OUT EX5) nil)
                ((IN1 O2) (OUT A4) nil)
                ((IN2 O2) (OUT A3) nil)
                ((IN2 A4) (OUT O1) nil)
                ((IN2 EX6) (OUT O1) nil)
                ((OUT EX6) 1 nil)
                ((OUT O2) 1 nil)
    The definitions for the Boolean functions are also included as
    demodulators.
```

Figure 12. Lisp input to the theorem prover for the two bit adder-subtracter

symbol *nil*. This symbol indicates that the demodulator was part of the input and has no ancestors.

As implemented, the inference mechanism is not complete. The lack of completeness manifests itself when the inference mechanism is called to label deep nodes in the hitting set tree. (A deep node is a node with relatively many components labeling the edges on the path from the root to the node. Recall that these components are treated as functioning abnormally.) Two points should be noted. First, none of the diagnosticians based on first principles implemented thus far are complete as implemented. While the diagnostic approach is theoretically complete, the inference mechanisms are not complete. Second, a loss of completeness at deep nodes is not as serious a problem as it might first appear. The deeper the node is in the HS-tree, the more components are involved in the potential diagnosis. Generally, a diagnosis containing several components is less likely than a diagnosis which involves only one or two components.

The intention was to build a prototype and then establish completeness by building in more equality reasoning and further specializing the inference mechanism by including focusing techniques appropriate to the domain of circuits. However, the initial results did not indicate that this type of inference mechanism would perform efficiently enough to be of use for the diagnostic problem. Results for the two example circuits are shown in Table II.

Certainly, part of the inefficiency of the process is due to the use of *BN-unification* in the superposition of the rules. BN-unification is a form of associative-commutative unification and its use is necessary because of the associative and commutative properties of the operators $\oplus$ and $\wedge$. Even so, the performance of the theorem prover on the circuit problems was much worse than was expected based on the results in Hsiang's paper.

Table II. STATISTICS ON THE DETERMINATION OF CONFLICT SETS BY A TERM REWRITING THEOREM PROVER

Circuit	Conflict Set	Input Rules	New Rules	Time (sec)	Abnormal Components
Full adder	(EX1 EX2)	39	14	28.74	--
	(A1 A2 O1 EX1)	38	31	87.60	EX2
	(A2 O1 EX1)	37	22	54.01	EX2 A1
Adder-subtracter	(EX1 EX2 EX3 EX4 A2)	68	129	2482.50	--
	(EX1 EX2 EX3 A1)	67	64	593.57	A2
	(EX1 EX2 EX4 EX5 EX6 A1 A2 O1)	67	122	1720.58	EX3
	(EX1 EX4 EX5 EX6 A1 A2 A3 O1)	67	122	1626.01	EX2
	(EX1 EX2 EX3 EX4)	66	103	1414.47	A2 A1
	(EX1 EX4 EX5 EX6 A1 A2 O1)	66	101	1205.84	EX2 A3
	(EX1 EX2 EX3 A2)	66	64	577.58	EX4 A1
	(EX1 EX2 EX3)	65	58	498.32	EX4 A1 A2

However, on examples from Hsiang's paper the theorem prover performed efficiently and comparable results were obtained. This raises a significant research question in the area of automated reasoning. Researchers in automated reasoning have long pointed to the need to be able to determine from the characteristics of the problem to be solved what type of inference mechanism and techniques should be used. This is something that mathematicians seem to learn through experience. A

more specific research question raised by this work is to characterize the circumstances under which a theorem prover based on term rewriting and the complete set of reductions for Boolean algebra performs well.

C. CONSTRAINT PROPAGATION

When reasoning within a specific domain, it is common to use focusing techniques in conjunction with a general theorem prover. When one knows how the proof is likely to be found or where the proof will not be found, the use of weighting strategies or specific inference rules can be helpful in directing the theorem prover.

The use of constraint propagation as the inference mechanism is appropriate for the diagnostic domain. This can be seen by considering the source of an inconsistency in SD $\cup$ OBS $\cup$ $\{\neg AB(c) \mid c \in C \subseteq \text{COMPONENTS}\}$. If the system description is correct, then SD $\cup$ $\{\neg AB(c) \mid c \in \text{COMPONENTS}\}$ is consistent. An inconsistency in SD $\cup$ OBS $\cup$ $\{\neg AB(c) \mid c \in C \subseteq \text{COMPONENTS}\}$ will occur when a behavior predicted by SD $\cup$ $\{\neg AB(c) \mid c \in C \subseteq \text{COMPONENTS}\}$ differs from the observed behavior. Therefore, the emphasis should be placed on the determination of predicted values. If a weighting function is used, the function should attach more importance to unit equality clauses. These clauses will determine the value of the inputs and outputs of the components. A specialized inference rule, such as the UR-resolution rule discussed earlier, also emphasizes unit clauses.

When techniques such as weighting functions and special inference rules are used in this manner, the goal is to guide the inference mechanism so that in some ways it mimics what a human would do in the given situation. When given a diagram of a circuit or other device and values for inputs and outputs, a person would use the known values to determine the unknown, internal values in the circuit. The new values would be calculated by applying the laws or rules which govern the correct

operation of the components. When all components are assumed to be functioning correctly, the reasoning can be done in a "forward" fashion with the outputs of the components determined by inputs. This represents simulation of the operation of the device.

When one or more components is assumed to be functioning abnormally, the reasoning task becomes more complex and simulation is no longer sufficient for discovering contradictions. The output of an abnormally functioning component cannot be predicted from its inputs. However, there may be enough constraints on the operation of the other components in the device that the outputs of the malfunctioning component may be predicted from the behavior and interconnections of the other components. This may require that inferences be made about the value of inputs based on output values of a functioning component. Thus, the reasoning proceeds in both a forward (outputs determined from inputs by simulation) and backward (inputs determined from outputs by inferences) fashion. Note that the possibility exists that neither forward nor backward reasoning will be able to determine all of the values in the device.

The preceding discussion represents an informal description of constraint propagation ([*SS*77], [*DS*80]). The basis of the method lies in the determination of a new value (either an input or output of a component) whenever enough information about other inputs and outputs is known. The connections between components, which in this work are assumed to be ideal, function as conduits of values. When a new value is determined, the components which led to the determination of the value are recorded along with the value. These components are referred to as the antecedants of the value. When a contradiction is found, the antecedants determine the components in the conflict set.

The value of a particular input or output of a component can usually be determined in more than one way. When two values are calculated for the same input or output, a coincidence [*DS*80] is said to have occurred. The two values may be identical, in which case nothing need be done. The two values may be consistent but establish a new constraint. For example, the output of a component may at one point be determined to be some variable value represented as x_1. The same output might be determined to be 0 by another set of constraints. The coincidence is consistent and also establishes that $x_1 = 0$. The third possibility is that the values disagree which means that an inconsistency has been found. The antecedants of the inconsistency are the union of the antecedants of the two values.

When using constraint propagation as an inference mechanism, the choice of what value to propagate next is usually controlled by a queue. The queue may be structured in a first-in-first-out manner or the values in the queue may be ordered in some way. For example, if the size of the antecedant set is significant, the queue is ordered on that basis. This is the case in diagnosis when the inference mechanism must return minimal conflict sets.

IV. ANALYSIS OF HITTING SET TREE APPROACH

A. INTRODUCTION

As mentioned in the review of Reiter's work in Chapter II, a subtle error in the algorithm for constructing a pruned HS-tree can result in the loss of minimal hitting sets. An analysis of the error, a correction to the algorithm, and a discussion of the effectiveness of pruning in reducing the number of calls to the inference mechanism are presented in this chapter.

A crucial component of a first principles diagnostician is the inference mechanism. Theoretical bounds are developed for the number of calls which must be made to the inference mechanism to determine the diagnoses and the empirical results of the implementation are compared to these bounds.

B. ANALYSIS OF PRUNING

One of the strengths of using Reiter's theory and the hitting set tree structure as the basis of a diagnostician is that the conflict sets returned by the inference mechanism need not be minimal. Thus, it is likely that the inference mechanism can be kept simpler than is the case if minimality must be guaranteed. However, Reiter's techniques for handling the non-minimal conflict sets can result in an incomplete diagnostician as it is possible to prune diagnoses from the tree.

Recall the algorithm for generating a pruned hitting set tree for a collection of sets C. Reiter proposes that when a node n' is labeled by a set S' and there exists a node n labeled by a set S such that $S' \subset S$, the redundant edges emanating from node n and their associated subtrees should be removed. The redundant edges are those edges which are labeled by the elements of $S - S'$. Reiter considers these edges

redundant since the minimal hitting sets for C and the minimal hitting sets for $C - \{S\}$ are the same. While this is true, removing redundant edges is analagous to the "cut" operator in Prolog [*CM*81]. Just as the "cut" can result in a loss of completeness by pruning solutions, removing redundant edges can result in minimal hitting sets being lost.

The problem arises from the interaction of the pruning rule and the closing rules. A closing rule will close the node n when it finds another node n' which will lead to the same minimal hitting set(s). This, of course, assumes that the node n' will remain in the HS-tree. The pruning rule, however, may remove the node n', meaning that the path to any potential hitting sets will be totally lost, lost from the node n path when node n was closed and lost from the node n' path when node n' was pruned.

Consider the following example. Figure 13 shows an HS-tree for the collection of sets $C = \{\{a\,b\,c\}\ \{b\,d\}\ \{f\,a\}\ \{b\,c\,e\}\ \{b\}\}$. The tree was generated according to the method for generating a pruned HS-tree except that redundant edges were not removed. The minimal hitting sets are $\{a\,b\}$ and $\{b\,f\}$. Figure 14 shows the same tree except that when the set $\{b\}$ is discovered, the redundant edges and their subtrees emanating from the node labeled by the set $\{a\,b\,c\}$ are removed. This reduces the tree to the one shown in Figure 15. Note that the minimal hitting set $\{a\,b\}$ is not contained in this tree because the node under the a branch of the node labeled $(f\,a)$ has been closed.

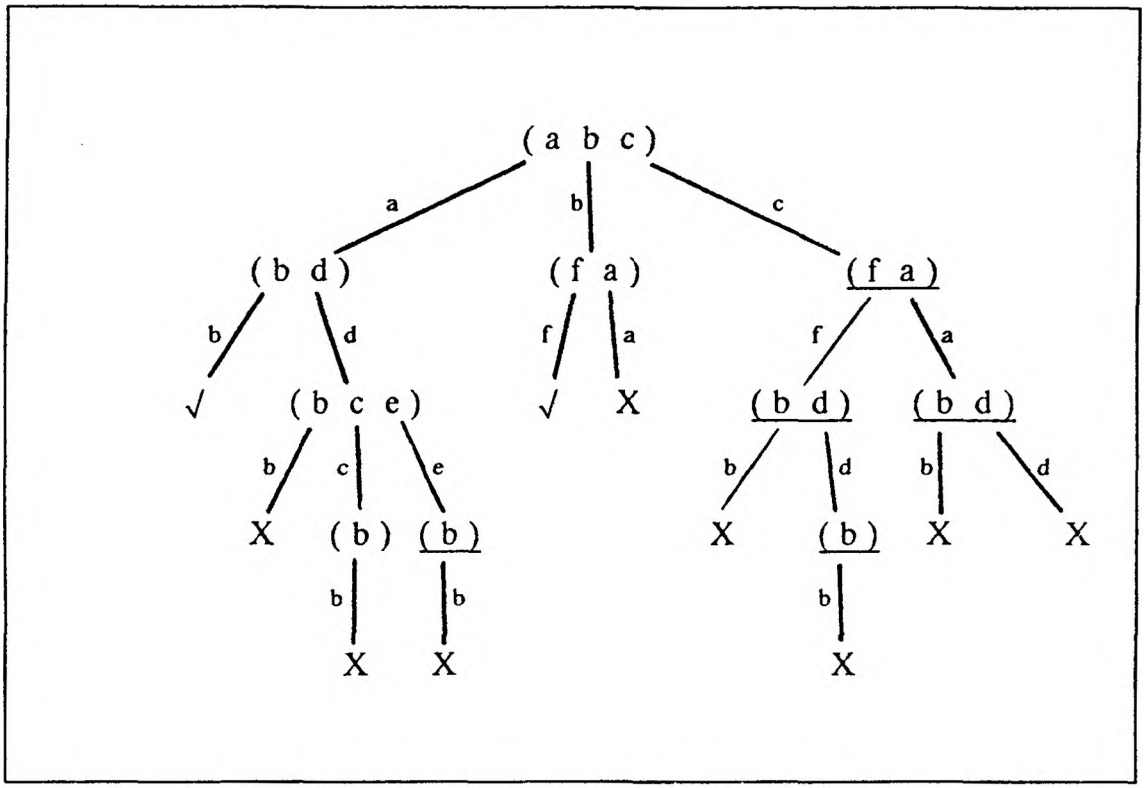

Figure 13. HS-tree without redundant edge pruning

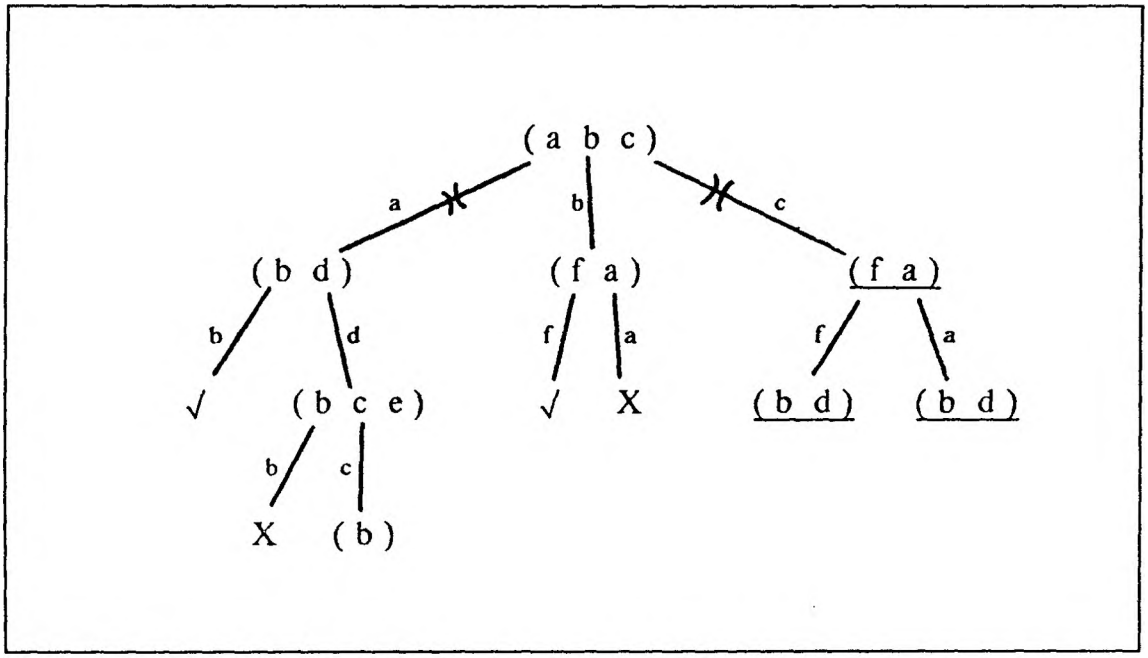

Figure 14. HS-tree with redundant edges marked

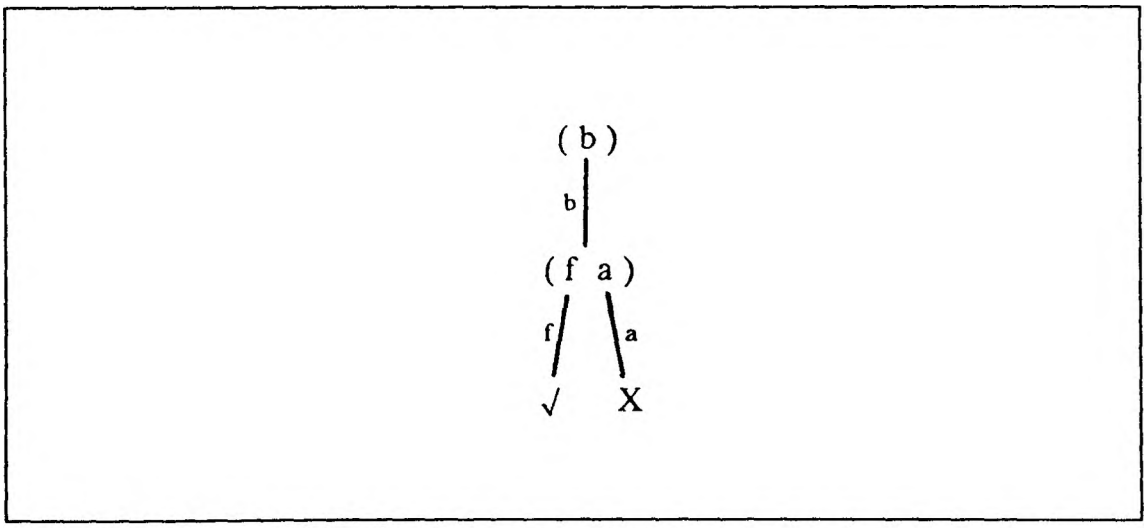

Figure 15. HS-tree with redundant edges and subtrees removed

In addition to the potential loss of minimal hitting sets, the removal of the so-called redundant edges can also lead to an increase in the number of calls to the underlying theorem prover. This can occur when there is a set labeling a node in the removed subtree which is not represented anywhere else in the tree. The set may have to be recomputed through a call to the theorem prover in order to label some node *m*. However, had the subtree not been pruned, node *m* could have made use of the existing label and a duplicate call to the theorem prover would not have been necessary.

It should be noted that it is possible for the removal of redundant edges to reduce the number of calls to the theorem prover. However, in numerous test cases using the implementation of this work, the number of calls has never been decreased by removing these edges and their associated subtrees. The following theorem will help to characterize when this type of pruning will decrease the number of invocations of the inference mechanism.

Theorem 1: Let S be a collection of sets which are minimal with respect to set inclusion. That is, for all distinct $X \in S$ and $Y \in S$, $X \not\subseteq Y$ and $Y \not\subseteq X$. If T is a hitting set tree for S, then for all $X \in S$, X labels a node in T.

Proof. Suppose that there exists an $X \in S$ that does not appear in T. Let X_0 be the root node of T. For each $x \in X_0$, there is an edge emanating from the root. Consider the edge labeled x_0 where $x_0 \notin X$. Such an x_0 exists, otherwise $X_0 \subseteq X$, which violates the condition that the sets are minimal with respect to set inclusion. Further, arrange the elements in X_0 so that x_0 is the first element in the set. Now, $X \cap \{x_0\} = \{\}$, so X is a possible label for the node which is connected to the root by the edge labeled x_0. However, since X is not in the tree, some other set X_1 was chosen to label the node. But there is an $x_1 \in X_1$ such that $x_1 \notin X$ by minimality with respect to set inclusion and $x_1 \neq x_0$ since $X_1 \cap \{x_0\} = \{\}$ by definition of hitting set tree.

Again, arrange the elements in the node label X_1 so that x_1 is the first element in the set. Let $X_2 \neq X$ label the node connected by the edge labeled x_1 to the node labeled X_1. Continue building the hitting set tree in this manner. Since the tree is finite, the process will terminate.

Let node n be the last node on the path labeled by a set of S and let X_n be the node label. $H(n) = \{x_0, \ldots, x_{n-1}\}$ and $H(n) \cap X = \{\}$ because of the manner in which the tree was constructed. Let node n' be the leaf connected to node n via the edge labeled x_n where $x_n \notin X$. Again, such an x_n exists. Two cases arise.

i) Node n' is labeled $\surd$. $H(n') = \{x_0, \ldots, x_{n-1}\} \cup \{x_n\}$. However, $X \cap \{\{x_0, \ldots, x_{n-1}\} \cup \{x_n\}\} = \{\}$, so X is a candidate label for node n' and therefore, node n' cannot be labeled $\surd$.

ii) Node n' is closed. Two cases arise here.

1) There exists a node m labeled $\surd$ such that $H(m) \subseteq H(n') = \{x_0, \ldots, x_n\}$. But, $X \cap \{x_0, \ldots, x_n\} = \{\}$ so $X \cap H(m) = \{\}$ and X is a candidate for labeling node m. Therefore, node m cannot exist.

2) For some node m, $H(m) = H(n')$. However, the elements of the sets $X_0, \ldots, X_n$ were always arranged so that the $x_0, \ldots, x_n$ were first in each. Therefore, this path is the left-most path in the tree and the node n' is the first node labeled at this level in the tree. Thus, there can be no such node m.

Since these cases are exhaustive and a contradiction was found in each case, there exists a node in the hitting set tree T which is labeled by the set X. □

Corollary to Theorem 1: Let S be a collection of sets which are not necessarily minimal with respect to set inclusion. Let $S' \subseteq S$ be the collection of all $X \in S$ such that X is not a superset of any other set in S. Thus, the collection of sets S' is

minimal with respect to set inclusion. If T is a hitting set tree for S, then for all $X \in S'$, the set X labels some node in T.

Proof: Suppose that there exists some $X \in S'$ such that X does not label a node in T. Consider the construction of T as in the proof of Theorem 1. Since X is not a superset of any other set in S, for all $X_i \in S$ there exists an $x_i \in X_i$ such that $x_i \notin X$. Therefore, the proof from Theorem 1 holds. □

Since all of the minimal sets in the collection must label a node of the hitting set tree T, in the diagnostic setting all of the minimal conflict sets for (SD,COMPONENTS,OBS) will eventually have to be found by the inference mechanism and will appear in the hitting set tree.

For any set $c' \in C$ such that c' is a superset of some set in the collection, whether c' labels a node depends on the order in which the sets are discovered. Because node labels are reused whenever possible, a superset can never be computed as a label when its subset is already known.

Consider the following example. Figures 16 and 17 show HS-trees for the collection of sets $\{\{c\,a\}\ \{a\,d\}\ \{d\}\ \{b\,a\}\ \{b\,c\}\ \{e\,b\}\ \{a\}\ \{e\}\}$. In the tree in Figure 16, the redundant edges are not removed and ten references to the collection of sets (calls to TP) are required in order to build the tree. In Figure 17, the redundant edges and subtrees are removed and the reference to the collection of sets which returned the set $\{e\,b\}$ is eliminated. It should be noted that this example was constructed by hand and in all of the diagnostic problems studied, the number of calls to TP has never decreased when the redundant edges are removed.

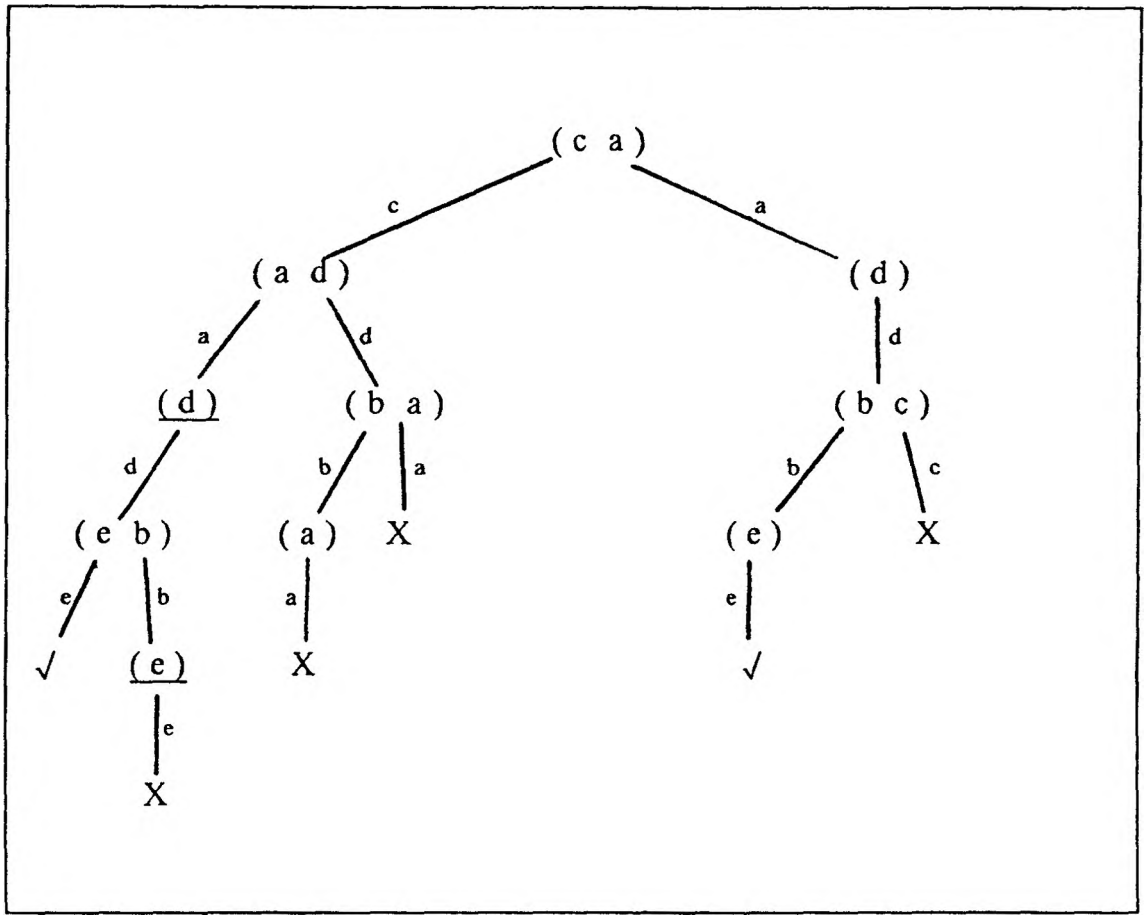

Figure 16. HS-tree to illustrate number of calls to TP when pruning is not used

By the Corollary to Theorem 1 all of the minimal sets in the collection will appear in the tree. Therefore, removing redundant edges and their associated subtrees will decrease the number of calls to the theorem prover only if all three of the following conditions hold:

1) Suppose that the redundant subtree is not removed and it contains a node n' which is labeled through a call to TP after the subtree is found to be redundant. Let c' be the label for node n'. (This is the node labeled (*e b*) in the example.)

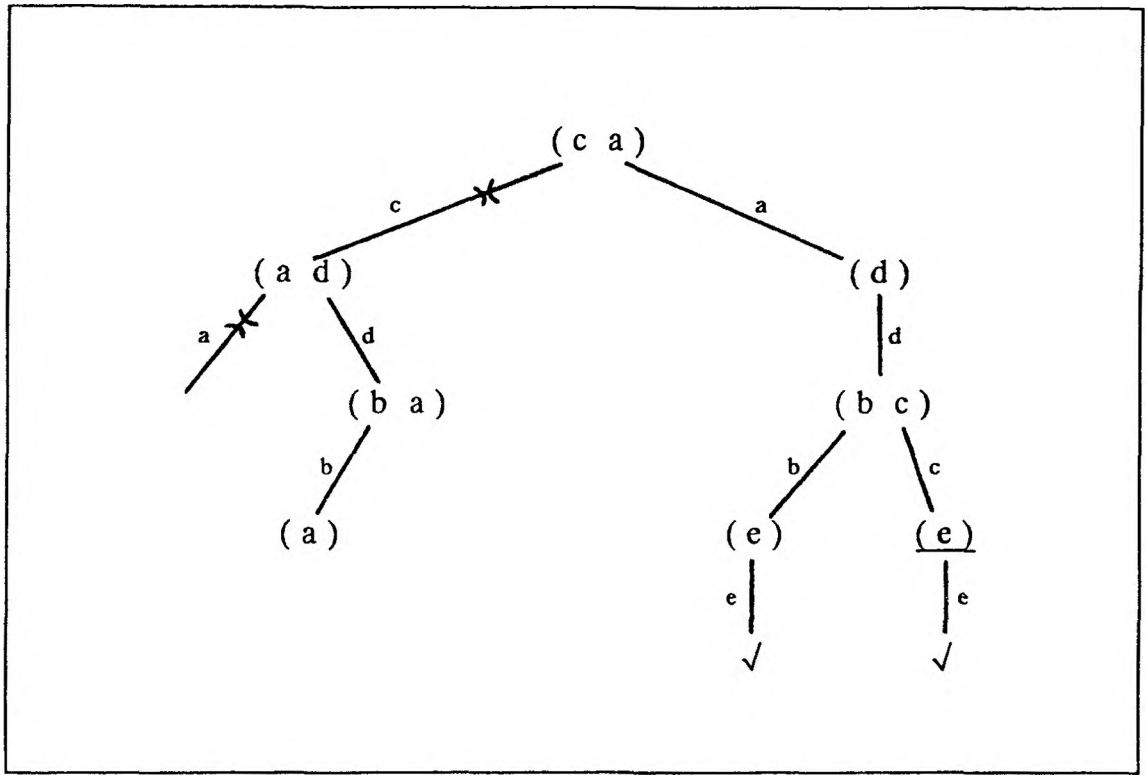

Figure 17. HS-tree to illustrate number of calls to TP when pruning is used

2) After n' is labeled, a node n outside of the redundant subtree is labeled by a call to TP. The set c which labels node n is a proper subset of c'. (This is the node labeled by the set (e) which is not underlined in the example.)

3) The tree has to be such that if the redundant subtree is removed, the set c' is not returned by TP as the label for some other node before the set c is returned as the label for node n. This would mean that the call to TP which returned the superset c' is eliminated by the pruning.

The advantages to removing the redundant subtrees are that the size of the HS-tree is reduced and potentially fewer calls are necessary to construct the HS-tree. However, the construction algorithm must be modified to ensure that all minimal

hitting sets are found and to keep from making extra calls to the theorem prover. When a redundant edge and associated subtree are to be removed, it is necessary to:

> 1) Reopen any node n' which was closed on the basis of $H(n') = H(n)$ where n is a node in the redundant subtree being pruned. This will prevent the loss of any minimal hitting sets.
>
> 2) Save any node labels which do not occur elsewhere in the tree. Such a label may be reused and thus an unnecessary call to the underlying theorem prover can be avoided.

The difficulty with these modifications is that they require a significant amount of overhead and are not straightforward to implement. When a closed node is reopened, the subtree rooted at that node must be expanded to the same depth as the rest of the HS-tree before the breadth-first construction of the tree can be continued. Also, the node labels which are saved for possible reuse must be maintained in a separate list and that list must be checked whenever a new node is to be labeled. As discussed in the next section, the inference mechanism used in this work returns conflict sets which are close to minimal. Thus, no benefit has been gained through pruning and it has been removed from the system.

The difficulties associated with pruning can be overcome through the use of a directed acyclic graph instead of the tree structure. This is discussed in more detail in [GS88]. Reiter presents the definition of the HS-tree and then develops the algorithm to build a pruned HS-tree. That pattern is followed here with respect to the construction of an HS-*dag* (directed acyclic graph). Reiter did not present a proof of correctness for the pruned HS-tree construction algorithm. Such a proof for the HS-*dag* is presented in [GS88].

Let D represent the growing HS-*dag* for an ordered collection of sets, C. Note that Reiter does not order the collection of sets used by the HS-tree algorithm. An ordering can be imposed, however, without loss of generality. An HS-*dag*, D, for C is constructed as follows.

1) Select the first set in C as the label for the root of the HS-*dag*. If C is empty, label the root by $\surd$.

2) Process the nodes in D in a breadth-first order. To process a node n

i) Define $H(n)$ to be the set of edge labels on the path in D from the root to node n.

ii) If for all $c \in C$, $c \cap H(n) \neq \{\}$ then label the node n by $\surd$. Otherwise, let Σ be the first member of C such that $\Sigma \cap H(n) = \{\}$. Note that by selecting Σ to be the first such member of C the algorithm will reuse node labels whenever possible.

iii) If node n is labeled by a set Σ then for each $\sigma \in \Sigma$ generate a downward arc from node n and label the arc by σ. The node to which this arc leads will be processed after all nodes belonging to the same generation as node n have been processed.

The above represents the directed acyclic graph equivalent of Reiter's basic definition of a hitting set tree without pruning. Pruning by closing nodes and removing redundant subgraphs is added to the construction of the HS-*dag* according to the following. Note that the unintended interaction of closing nodes and pruning which resulted in the loss of minimal hitting sets from the HS-tree cannot occur in the HS-*dag*.

1) Let n be the node being processed. The algorithm will not always generate a new node m as the descendant of node n. There are two cases to consider:

i) If there is a node n' in D such that $H(n') = H(n) \cup \sigma$, then let the arc under n which is labeled by σ point to the existing node n'. Hence node n' will have more than one parent. In a pruned HS-tree node n' would have been closed.

ii) If the downward arc from node n labeled by σ cannot point to an existing node, generate a new node m at the end of the σ-arc as described in the basic HS-*dag* algorithm.

2) If the set Σ is to label a node and it has not been used previously, then attempt to prune D.

i) If there is a node n which has been labeled by a set S of C such that $\Sigma \subset S$, then relabel n with Σ.

ii) For any $\alpha \in S - \Sigma$ the edge under node n which is labeled by α is no longer allowed. In the subgraph connected to node n by the edge labeled α delete all nodes except those which have an ancestor which is not being removed. Note that this may eliminate the n which is currently being processed.

iii) As discussed in Reiter's description of the HS-tree, the minimal hitting sets for the collection C are the same as the minimal hitting sets for $C - S$.

Therefore, the set S should not be used to label any other nodes in D and should be removed from C.

3) If there is a node n which is labeled by $\surd$ and whose $H(n)$ is a strict subset of $H(n')$ for some node n' then close the node n'. A label is not computed for n' nor are any successor nodes generated. A closed node is denoted by $\times$.

As is the case with the HS-tree, the particular HS-*dag* which the algorithm constructs depends on the order of the sets in the collection. If $\Pi(C)$ refers to the Π rearrangement of the sets in C, the HS-*dag* for C and the HS-*dag* for $\Pi(C)$ may be different but the minimal hitting sets determined by the two directed cyclic graphs will be the same.

Consider again the collection of sets which was used earlier to illustrate the problem with pruning the HS-tree. This is the collection of sets $C = \{\{a\ b\ c\}\ \{b\ d\}\ \{f\ a\}\ \{b\ c\ e\}\ \{b\}\}$. The unpruned HS-tree for C is shown in Figure 13. Figure 18 shows the HS-*dag* constructed up to the point where the set $\{b\}$ is first used to label a node. Since this is the first use of the set $\{b\}$ and it is a proper subset of a set already labeling a node (in fact, several nodes are labeled by sets which are supersets of $\{b\}$), pruning as described in Step 2 of the pruning algorithm is to be applied.

In order to simplify the discussion, let the root node be pruned first. After these redundant edges and subgraphs connected to the root are removed, no other pruning will be necessary. However, it should be noted that the order in which the redundant edges are pruned is irrelevant. Ultimately, the same result would have been attained had a node other than the root node been pruned first.

The pruning is done by first relabeling the root by the set $\{b\}$. The subgraphs which are connected to the root by the edges labeled a and c are no longer valid. Thus, these subgraphs are removed, except that any node which has an ancestor that is not being removed remains in the HS-*dag*. In this example, the node labeled $\surd$

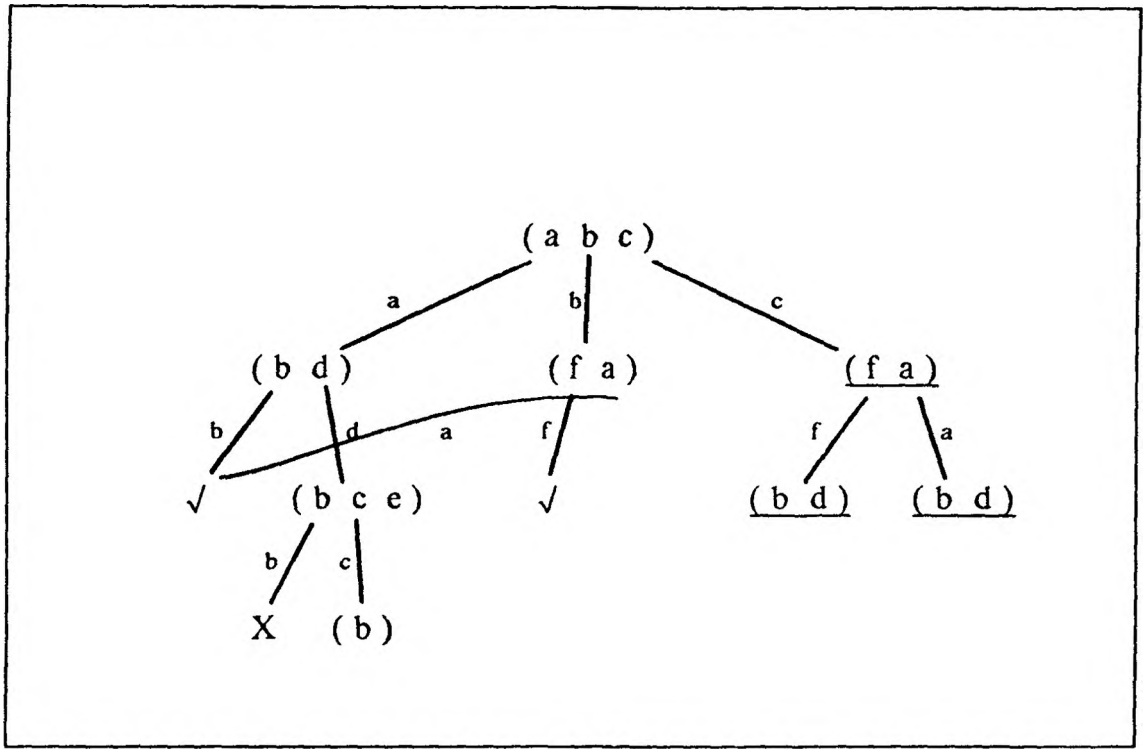

Figure 18. Partial HS-dag for a collection of sets

which corresponds to the minimal hitting set $\{a\,b\}$ remains since not all of its ancestors are removed. After the HS-*dag* is pruned, any set which is a strict superset of the set $\{b\}$ is removed from the collection. Thus, C becomes $\{\{f\,a\}\ \{b\}\}$. The pruned HS-*dag* constructed by the algorithm is shown in Figure 19. Note that both of the minimal hitting sets, $\{a\,b\}$ and $\{b\,f\}$, are in the HS-*dag*.

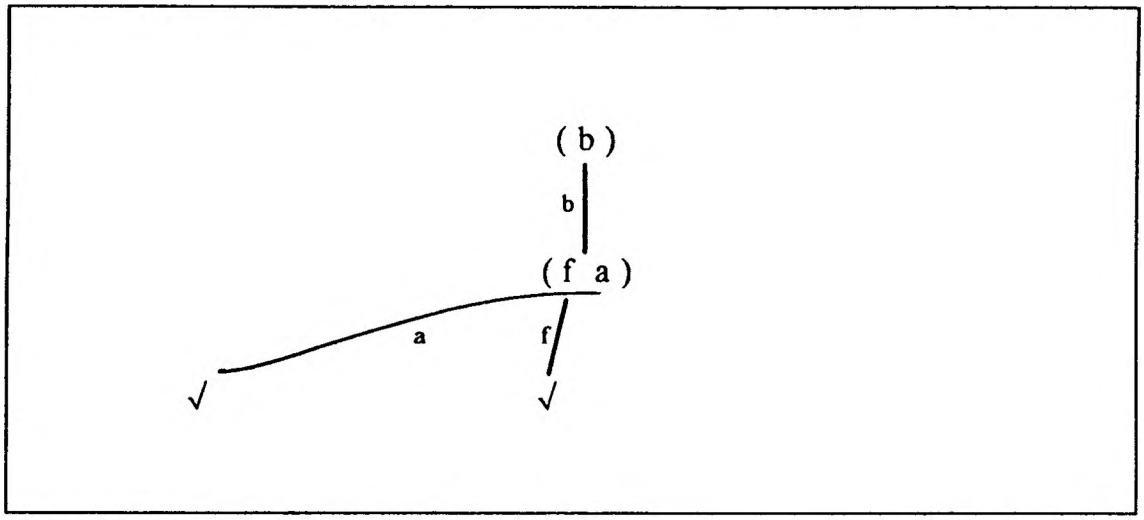

Figure 19. Pruned HS-dag

C. COMPLEXITY ANALYSIS AND PERFORMANCE RESULTS OF IMPLEMENTATION

In order to evaluate the performance of the program which implements the diagnostic theory, it is necessary to establish theoretical bounds for the best case performance and then compare these to the results of the implementation. The heart of the diagnostic algorithm is the underlying theorem prover. Thus, the theoretical bounds will be determined based on the number of calls to the inference mechanism which are necessary to determine the set of diagnoses.

The HS-tree must contain all of the minimal conflict sets and the determination of a minimal conflict set requires an invocation of the inference mechanism. In addition, there must be a node labeled √ which corresponds to a diagnosis. Labeling a node by √ also requires an invocation of the inference mechanism. Thus, in the best case, the number of calls to the theorem prover is given by:

N = number of minimal conflict sets + number of diagnoses.

This quantity is bounded below by

N' = number of diagnoses + max { $|d|$ where d is a diagnosis}.

There is a large gap between the best case and worst case performance. In the worst case, the theorem prover returns the largest possible conflict set when an inconsistency is found. The maximum number of calls in any diagnostic session depends on: 1) the number of components in the system description, 2) the number of diagnoses for the observed behavior, and 3) the distribution of the diagnoses over the set of components. For example, consider a device which has 4 components, (c_1, c_2, c_3, c_4), and 2 diagnoses each consisting of 2 components. If the diagnoses are (c_1 c_2) and (c_3 c_4), the maximum number of calls to the theorem prover is 11. If the diagnoses are (c_1 c_2) and (c_1 c_3), the maximum number of calls is 12.

In considering the best case performance, the number of diagnoses is generally the larger contributor to the number of calls. In addition, calls to the theorem prover when SD $\cup$ OBS $\cup$ $\{\neg AB(c) \mid c \in C \subseteq COMPONENTS\}$ is consistent are usually more time consuming than those calls where an inconsistency is found. When the system description and observations are consistent, every possible path must be explored by the theorem prover in order to verify the consistency. When inconsistent, the theorem prover can return the conflict set as soon as the inconsistency is found. Therefore, where appropriate, methods such as halting construction of the HS-tree beyond a certain level and expanding the tree according to the probability of the diagnosis can significantly reduce the number of the most costly calls to the theorem prover by eliminating some calls which return $\surd$.

Reiter's theory allows the inference mechanism to return a conflict set which is not minimal. However, anytime a non-minimal set is returned, the computational effort of the call is, in essence, wasted. The advantage to allowing the inference

mechanism to return a non-minimal conflict set is that the inference mechanism need not employ ordering schemes in order to guarantee minimality. The most desirable course is to have the theorem prover return conflict sets that are as close to minimal as possible without significantly increasing the complexity of the theorem prover. The details of the implementation of the inference mechanism based on constraint propagation are presented in Chapter VI.

Table III shows statistics from representative example problems. Diagrams of the circuits, except for the full adder which is shown in Figure 1 in Chapter II, are presented in Chapter VI. The statistics were collected from runs which diagnosed the circuit under all possible combinations of input and output values for the circuit.

Table III. STATISTICS ON THE PERFORMANCE OF THE DIAGNOSTICIAN

Circuit	Number of Runs	Total time (seconds)	Conflict Sets Returned	Minimal Conflict Sets
Full adder Figure 1	32	63.88	39	39
Two bit adder Figure 29	128	592.48	286	252
Overflow detector Figure 30	256	8767.06	1096	661
Adder-subtracter Figure 31	256	10690.15	1268	962
BCD-binary Figure 32	256	6686.19	868	680
TOTAL	928	26799.76	3557	2594

Two points should be noted concerning the data in Table III. First, the times given are CPU times for the entire diagnostic process over all possible input and output values. Therefore, they represent a significant improvement over the times presented in Tables I and II for the determination of conflict sets by ITP and the theorem prover based on term rewriting. Second, 73% (2594 of 3557) of the conflict sets returned by the inference mechanism are minimal. Thus, despite the fact that the control structure used within the implementation of the diagnostician is relatively simple, the conflict sets being returned are close to minimal.

V. EXTENSIONS TO THE THEORY OF DIAGNOSIS

A. INTRODUCTION

Several important research questions arise with respect to the general problem of diagnosis as well as the specific approach of diagnosis from first principles. In this chapter, some of these questions are addressed through extensions to Reiter's theory. These are: the determination of the diagnosis when there are multiple sets of observations, using new measurement information in conjunction with the existing hitting set tree, and a heuristic for suggesting measurements to decrease the number of diagnoses.

B. MULTIPLE SETS OF OBSERVATIONS

It is often the case that diagnoses are to be found for a system and those diagnoses must explain the system behavior under several different sets of input values. Let $OBS_1, OBS_2, \ldots, OBS_n$ be sets of observations. Different situations can arise when multiple observations are available and a method for finding the diagnoses in each of these situations is necessary. Three cases can be identified:

1) All of the OBS_i are available from the start of the diagnostic process. Although Reiter does not discuss multiple observations and it is not apparent from his presentation, the HS-tree algorithm is applicable in this case providing that an appropriate representation and underlying theorem prover are available.
2) An HS-tree has been constructed for $OBS_1, \ldots, OBS_m$ and new observations $OBS_{m+1}, \ldots, OBS_n$ become available. This could be handled under case 1) with a new HS-tree for $OBS_1, \ldots, OBS_m \cup OBS_{m+1}, \ldots, OBS_n$ constructed. However, in some cases this will result in a significant duplication of work. A method for augmenting the existing HS-tree to account for the new observations is needed.

3) The third possibility is that the device is diagnosed under each of the $OBS_1, \ldots, OBS_n$ individually yielding the diagnosis sets $D_1, \ldots, D_n$. The composite diagnosis Δ can be determined from the individual diagnoses $D_1, \ldots, D_n$. This set Δ is the set of minimal hitting sets for $D_1, \ldots, D_n$.

Each of these cases is discussed in the following sections.

One of the benefits of having multiple sets of observations available is that more information is available about how the system is functioning. This information can decrease the number of possible diagnoses. This is usually the case if the non-intermittency assumption is applied.

The non-intermittency assumption is frequently applied in diagnostic work, particularly in the case of circuit diagnosis. It requires that all components, even those which are faulty, behave consistently over time. Thus, if the inputs to some component at times t_i and t_j are the same, under the non-intermittency assumption, the output values of the component are also the same. As Genesereth [*Ge*84] points out, the assumption is not valid when applied to long intervals of time since it would imply that no component would ever fail. However, when dealing with a short period of time, such as the time between observations in a diagnostic session, it may be a reasonable assumption.

A comprehensive theory of diagnosis from first principles should not be based on assumptions such as the single fault assumption or the non-intermittency assumption. These assumptions, however, are commonly applied so they must fit into the theory. Reiter addresses the single fault assumption and points out that the single fault diagnoses are precisely those diagnoses which occur at level 1 in the HS-tree. No mention is made of the non-intermittency assumption.

Building the non-intermittency assumption into an automated diagnostician requires that the assumption be implemented in the model and that the underlying theorem prover be able to apply the assumption. In the DART system discussed earlier [*Ge*84], the user can include or exclude the assumption as appropriate for the domain of the diagnostic problem.

A difficulty that arises in implementing the non-intermittency assumption is that it requires that the underlying theorem prover be more complex. Information concerning the input and output values of the components must be maintained and this information must then be used to determine if the non-intermittency assumption is violated by a result. If so, the line of reasoning which led to the violation must be rejected. The relationship of the non-intermittency assumption to the various methods for handling multiple sets of observations must be considered.

Initially, Reiter's definition of a diagnosis does not appear applicable when there are multiple observations. The definition is applicable, but what is not discussed is the information which must be represented within the model and the type of reasoning of which the theorem prover must be capable in order to compute the diagnosis. The representation and underlying theorem prover must make use of a time indicator or an observation number. Suppose the following observed data for the full adder are available.

I:	II:
IN1(EX1)=0	IN1(EX1)=1
IN2(EX1)=1	IN2(EX1)=1
IN1(A2)=1	IN1(A2)=0
OUT1(EX2)=1	OUT1(EX2)=0
OUT1(O1)=0	OUT1(O1)=0

The representation must allow the theorem prover to handle these data separately but all of the observations must be explained by the same diagnosis. This can be accomplished by the use of a representation such as:

(IN1 EX1 t=1 0)
(IN1 EX1 t=2 1)

(IN2 EX1 t=1 1)
(IN2 EX1 t=2 1)
(IN1 A2 t=1 1)
(IN1 A2 t=2 0)
(OUT1 EX2 t=1 1)
(OUT1 EX2 t=2 0)
(OUT1 O1 t=1 0)
(OUT1 O1 t=1 0)

where the first two parameters identify the input/output descriptor of the component, the third parameter identifies the time or observation set, and the fourth parameter is the value.

Whatever the implementation of the underlying theorem prover, it must reason within the time or observation set classification to find a contradiction and the corresponding conflict set. The theorem prover must return $\surd$ when all time classifications are consistent. The techniques of closing a node and reusing node labels are applicable as before. The tree in Figure 20 is one of the HS-trees which could be built as a result of these data. The labels I and II identify the set of data for which the contradictions and resulting conflict set were found. Recall that an underlined node label is a label which could be determined from another label in the tree without making a call to the theorem prover.

For efficiency concerns, the theorem prover should be told when a set of observations has already been found to be consistent along some path in the HS-tree. This can avoid unnecessary computations, which is especially important since it generally requires more time to show a set of clauses consistent than to find an inconsistency. Suppose that SD $\cup$ $OBS_i \cup [\neg AB(c) \mid c \in COMPONENTS - \delta]$ has been found to be consistent at node n where $\delta = H(n)$. For any superset δ' of δ, $SD \cup OBS_i \cup [\neg AB(c) \mid c \in COMPONENTS - \delta']$ will also be consistent. Thus, in the general case, OBS_i need not be considered in the determination of the node labels of the descendants of node n. This is not the case, however, when the non-intermittency assumption is to be applied. In that case, all of the observations

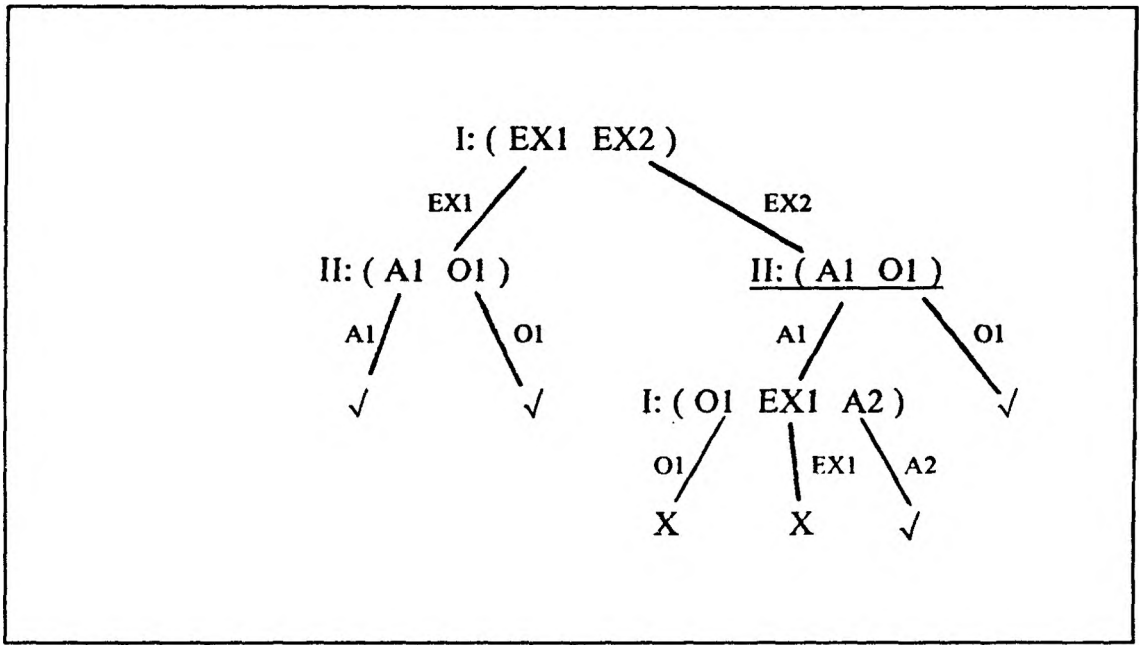

Figure 20. Diagnosis with multiple sets of observations used to build the HS-tree

must be considered simultaneously in order to verify that the behavior of each component is consistent over time.

If an HS-tree already exists for $OBS_1, \ldots, OBS_m$, it is possible to make use of that HS-tree as a starting point for adding new observations $OBS_{m+1}, \ldots, OBS_n$. The resulting HS-tree may yield a larger or smaller number of diagnoses. The following procedure results in an augmented tree which yields the diagnoses for SD $\cup$ $\{OBS_1, \ldots, OBS_m\}$ $\cup$ $\{OBS_{m+1}, \ldots, OBS_n\}$.

1. Begin with the node k which is closest to the root and labeled by $\surd$.

2. For $j = m+1, \ldots, n$ test the consistency of $OBS_j \cup SD \cup [\neg \mathrm{AB}(c) \mid c \in \mathrm{COMPONENTS} - H(k)]$. If inconsistent, relabel node k with the conflict set returned by the underlying theorem prover.

3. Continue checking nodes originally labeled $\surd$ and expanding relabeled nodes in a modified breadth-first fashion. Check all nodes labeled $\surd$ on a level before determining the labels of any new nodes on that level. The techniques of reusing node labels and closing nodes are to be applied as in the construction of the hitting set tree.

Note that it is not necessary to consider nodes which were closed when the original HS-tree was constructed. A node n which is closed has an $H(n)$ value which is a subset of the $H(n')$ value for some node n' where node n' is labeled $\surd$. Since any node labeled $\surd$ is expanded until its label is again $\surd$ or can be closed, $H(n)$ will still be a subset of some node which is labeled by $\surd$.

Suppose the HS-tree in Figure 21 for OBS_I is known and the additional observation set OBS_{II} becomes available. The observed values are given below.

I:	IN1(EX1)=0	II: IN1(EX1)=1
	IN2(EX1)=1	IN2(EX1)=1
	IN1(A2)=1	IN1(A2)=0
	OUT1(EX2)=1	OUT1(EX2)=0
	OUT1(O1)=0	OUT1(O1)=0

The tree which results from augmenting the HS-tree for OBS_I with the values from OBS_{II} is shown in Figure 22. The new conflict sets are enclosed in brackets. The superscript of a new label indicates the order in which the node was added to the tree.

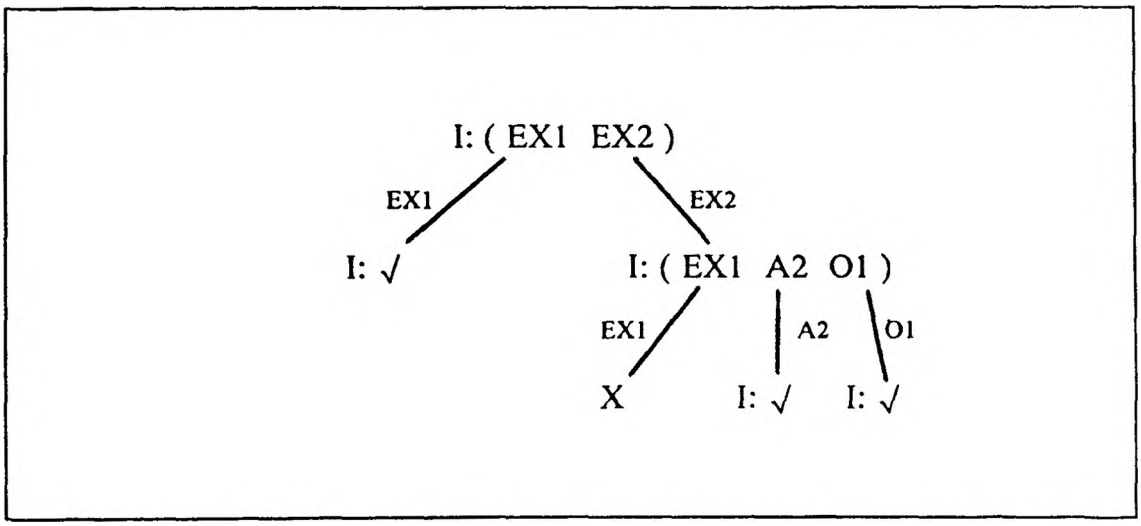

Figure 21. HS-tree for full adder under a single observation set

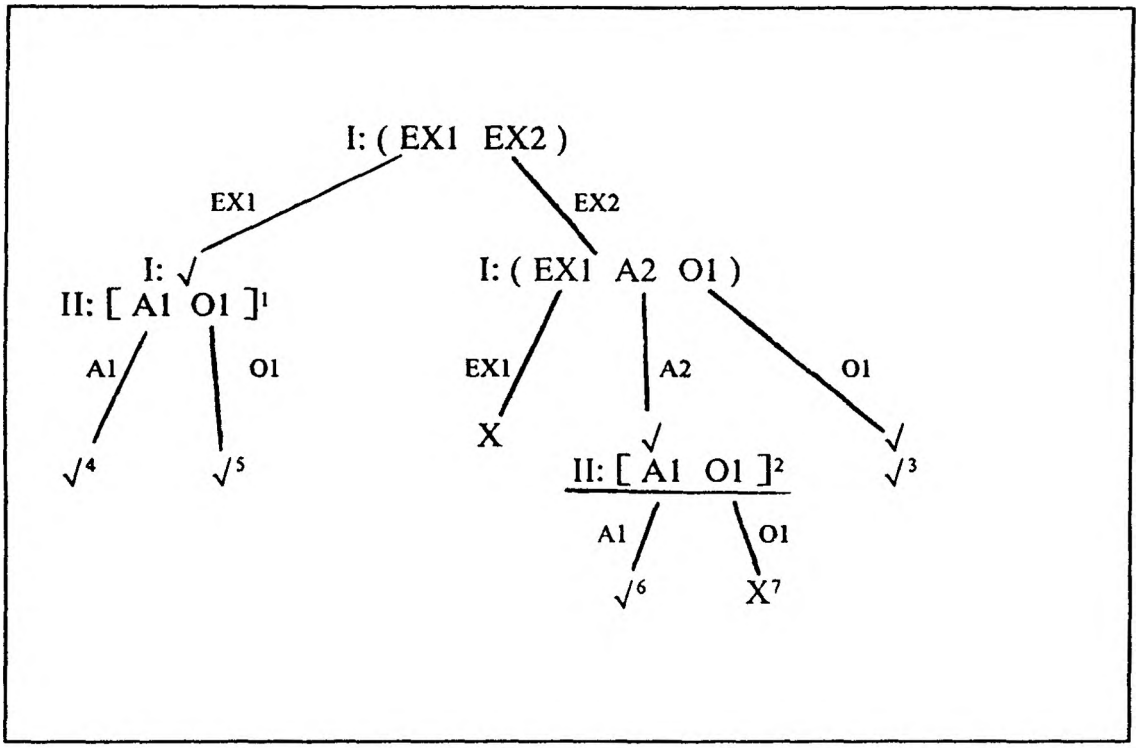

Figure 22. HS-tree augmented for a second observation set

A third means of determining the composite diagnosis Δ under the observations $OBS_1, \ldots, OBS_n$ is to determine the individual diagnoses $D_1, \ldots D_n$ and then create a DHS-tree. The DHS-tree will identify the collection of minimal hitting sets for $D_1, \ldots, D_n$ which is the composite diagnosis. The definition of a DHS-tree is similar to that of an HS-tree. Define a *DHS-tree* for $D_1, \ldots, D_n$ to be the smallest edge labeled and node labeled tree T with the following properties:

(1) If $D_1, \ldots, D_n$ are all empty, label the root by √. Otherwise, the root is labeled by one of the non-empty diagnosis sets $D_1, \ldots, D_n$.

(2) If n is a node of T, define $H(n)$ to be the union of the sets labeling the edges on the path in T from the root node to node n. If node n is labeled by √ then it has no successor nodes. If node n is labeled by a set D_i, then for each $d \in D_i$, node n has a successor node n_d joined to node n by an edge labeled by d. The label for the node n_d is a non-empty set D_j such that $\forall d' \in D_j$, $d' \notin H(n_d)$ if such a set of diagnoses exists. Otherwise, √ is the label for node n_d.

The same properties which hold for the HS-tree hold for the DHS-tree. If n is a node of the tree labeled by $\surd$, then $H(n)$ is a hitting set for $D_1, \ldots, D_n$. Also, each minimal hitting set for $D_1, \ldots, D_n$ is $H(n)$ for some node n of the the tree which is labeled by $\surd$.

The techniques for building a pruned HS-tree were designed to accomplish two goals: minimize the number of calls to the underlying theorem prover and generate only those hitting sets which are minimal. When the DHS-tree is built, no calls to the theorem prover are necessary since $D_1, \ldots, D_n$ are already known. Therefore, the technique of reusing node labels is not necessary, but it can be applied. However, closing nodes is necessary since only minimal hitting sets are desired.

In an HS-tree, the edges are labeled by a single value and the tree is built breadth-first. In a DHS-tree, the edges are labeled by sets. In order to generate the smallest tree with the above properties, the tree is not built in a strict breadth-first fashion. Rather, nodes are added in increasing size of the $H(n)$ set. This will guarantee that the nodes labeled by $\surd$ are the minimal hitting sets.

Suppose three sets of observations result in the individual diagnoses shown below. D_1, D_2, and D_3

D_1: {(A1) (O1)}
D_2: {(A1) (O1) (A2)}
D_3: {(EX1) (EX2 A1) (EX2 O1) (EX2 A2)}

A DHS-tree for D_1, D_2, and D_3 is shown in Figure 23. The order in which the nodes were labeled is indicated by the superscript of the node label. The composite diagnosis set is { (EX1 A1) (EX1 O1) (EX2 A1) (EX2 O1) }

The question which naturally arises as a result of this discussion is: Are there any advantages to using one of these methods of determining the composite diagnosis over the other two? The answer depends on whether all OBS_i are known initially or

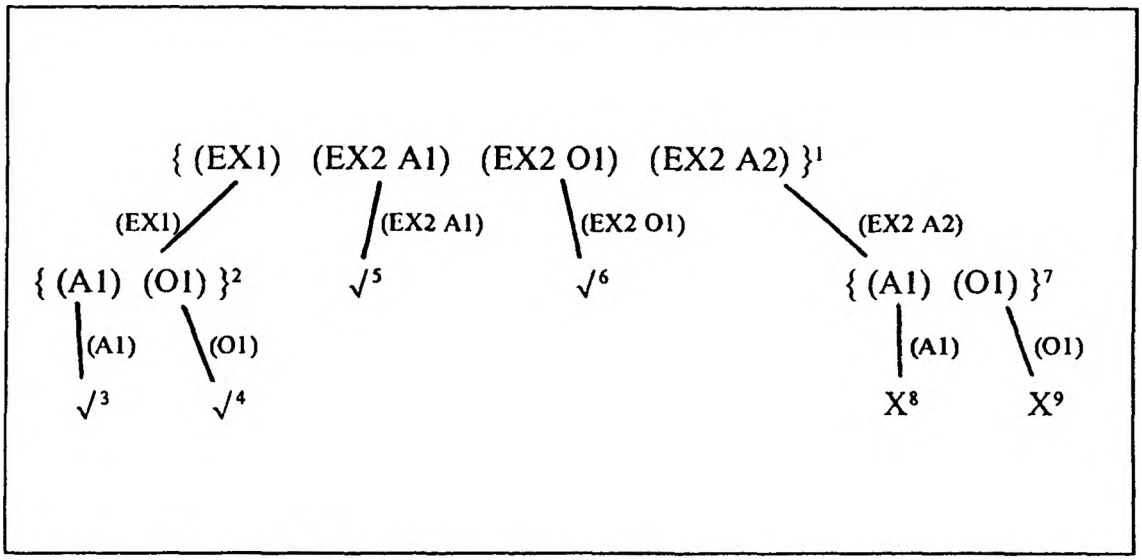

Figure 23. Example DHS-tree

are made known incrementally and is also tied to whether the non-intermittency assumption is to be applied.

If the non-intermittency assumption is to be applied, the composite diagnosis cannot be determined from the DHS-tree for $D_1, \ldots, D_n$. The individual diagnoses do not contain the necessary information concerning output values under a particular set of input values to determine whether a diagnosis violates the non-intermittency assumption. Thus, the use of the assumption requires that the composite diagnosis be determined by reasoning about all sets of observations at once, or that the required data values are stored as information associated with a node label so that it is available when the HS-tree is augmented.

The following example illustrates the effect that the non-intermittency assumption can have on the set of possible diagnoses. Suppose that two sets of observations are available for the full adder.

I:	IN1(EX1)=0 IN2(EX1)=0	II:	IN1(EX1)=0 IN2(EX1)=0

IN1(A2)=0	IN1(A2)=1
OUT1(EX2)=0	OUT1(EX2)=0
OUT1(O1)=1	OUT1(O1)=1

Without the non-intermittency assumption, the diagnosis set is { (EX1 A1) (EX1 O1) (EX1 A2) (EX2 A1) (EX2 O1) (EX2 A2) }.

Consider the three diagnoses involving the component EX1. Under both sets of observations, the inputs to EX1 are the same. If EX1 is faulty but functioning in a consistent manner, then its output in both cases is some value *c*. This value *c* is the value of the first input of EX2. Since EX2 is not included in any of the diagnoses which involve EX1, EX2 is assumed to be functioning correctly and two cases must be considered. In the first case, the inputs to EX2 are *c* and 0 resulting in an output of 0. Since EX2 is functioning correctly *c* must be 0. In the second case, the inputs to EX2 are *c* and 1 resulting in an output of 0. In this case *c* must be 1. There is no value for *c* which under the assumption of non-intermittency and the assumption that EX2 is not faulted that can explain this behavior. Thus, the non-intermittency assumption eliminates the three diagnoses involving EX1 and the diagnosis set becomes { (EX2 A1) (EX2 O1) (EX2 A2) }.

It should be noted that explanations for the system behavior other than those included in the above diagnosis set are possible. A short, for example, is a possible explanation. Additionally, the non-intermittency assumption may not be valid in this case. Considerations such as these are beyond the scope of the model, however.

C. USING THE HS-TREE WITH MEASUREMENT INFORMATION

In the diagnostic process, measurement information often becomes available after a hitting set tree has been built and a set of diagnoses have been determined for a system under an initial set of observations. The new measurement information will likely affect the set of possible diagnoses. In this section, a method for extending the original hitting set tree with the measurement information in order to obtain the new diagnoses is discussed. The method for extending the tree with measurement information is similar to that of extending the hitting set tree for multiple sets of observations.

Let T be an HS-tree generated for (SD,COMPONENTS,OBS) and Δ be the resulting set of diagnoses. Suppose that for all nodes which are labeled by $\surd$ the value of all of the components' inputs and outputs have been saved. Associated with each value is a list of components which led to the determination of that value. This list is referred to as the list of antecedants. These requirements are easily met when an inference mechanism based on constraint propagation is used. Let Π be a single measurement on the system, for example, OUT(EX1)=0. The method for handling such single measurements will be given. The extension for handling multiple simultaneous measurements is straightforward.

Recall Reiter's theorem concerning the effect of a measurement on the set of diagnoses. The theorem states that if a new measurement Π becomes available, those diagnoses which predicted Π will remain diagnoses. However, new diagnoses which are strict supersets of some of the diagnoses which predicted $\neg\Pi$ can arise. Using as much of the existing hitting set tree T as possible, the diagnoses in the presence of the measurement Π are to be determined. The method for expanding T is as follows:

1. Start with the node in T which is closest to the root and labeled $\surd$.

2. If node n is labeled $\surd$ and the measurement Π is consistent with the value predicted at node n, then $H(n)$ remains a diagnosis. Otherwise, relabel n with the antecedants of the inconsistent value.

3. Proceed in a modified breadth-first manner checking nodes and applying the techniques of reusing node labels and closing nodes. Check all nodes labeled $\surd$ on a level before adding any new nodes to the level. Label all nodes on level k before moving to level $k + 1$.

Two points should be noted. First, if the components' values and antecedants are saved when the tree is built, the consistency test of step 2 is simplified. If the values agree, a call to the underlying theorem prover is not necessary. Second, step 3 does not yield a strict left-to-right breadth-first expansion of the hitting set tree. Nodes on a level which are labeled $\surd$ are tested before any new labels are added on that level. This allows for the maximum use of the pruning techniques. Verifying that $H(n)$ for a node n is still a diagnosis may allow one or more of the descendants of a newly created node to be closed, thus reducing the number of calls to the inference mechanism.

The following example will illustrate the method for expanding the existing HS-tree to account for new measurement information. Consider the full adder under the set of observations used previously, namely,

IN1(EX1) = 1
IN2(EX1) = 0
IN1(A2) = 1
OUT1(EX2) = 1
OUT1(O1) = 0

An HS-tree for diagnosing the device is shown in Figure 24. Suppose that a measurement specifying that the output of component A2 is 1 is made available. If the input and output values of the components are saved when a node is labeled $\surd$, the computations necessary to extend the tree are minimized.

The node corresponding to the diagnosis (EX1) is checked first since it is the node labeled $\surd$ which is closest to the root. The value predicted at the node for the output of A2 is 0 which disagrees with the measured value. The component which

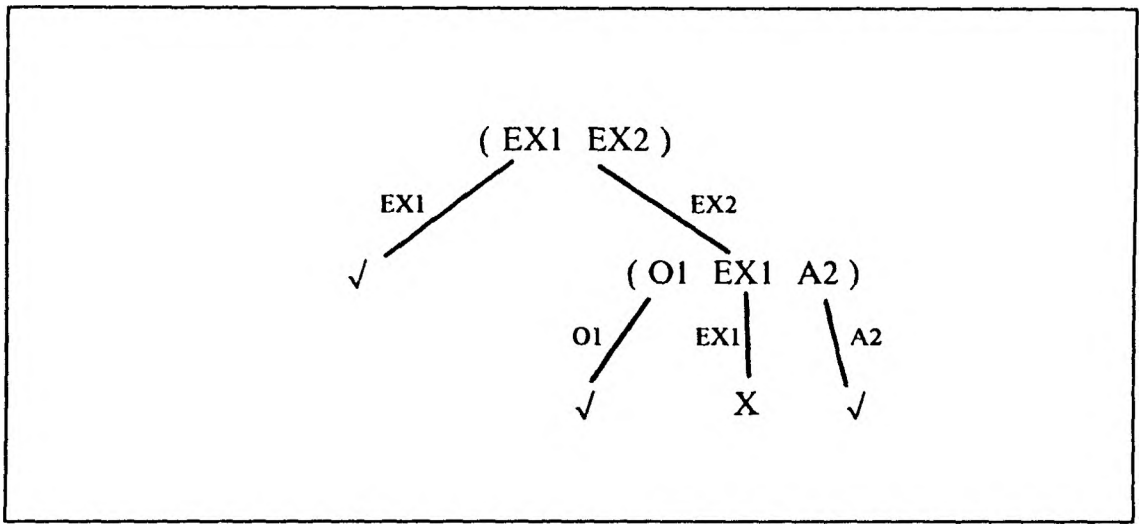

Figure 24. HS-tree for full adder before measurement information

determined this value is O1. Therefore, (O1) is a conflict set and becomes the new label for the node. The nodes labeled √ which correspond to the two double fault diagnoses are checked next. The value predicted for A2 at the node corresponding to the original diagnosis (EX2 O1) matches the measurement. Thus, (EX2 O1) remains a diagnosis. The predicted value for the output of A2 at the node corresponding to the original diagnosis (EX2 A2) need not be checked. There is no intersection between $H(n)$ for the node and the conflict set (O1), so the node is relabeled by this conflict set. The label for the descendant of the node with the superscript 1 is determined next. This is done through a call to the inference mechanism, under the assumption that components O1 and EX1 are not functioning correctly. The conflict set (A2 EX2) is returned. The expansion of the tree continues according to the algorithm for constructing a pruned HS-tree.

The augmented tree is shown in Figure 25. The new label of a node originally labeled √ is shown below the original label. New conflict sets are enclosed in brackets. The superscripts of the node labels show the order in which the labels were

assigned. The diagnoses which reflect the measurement are (EX1 A2 O1) and (EX2 O1).

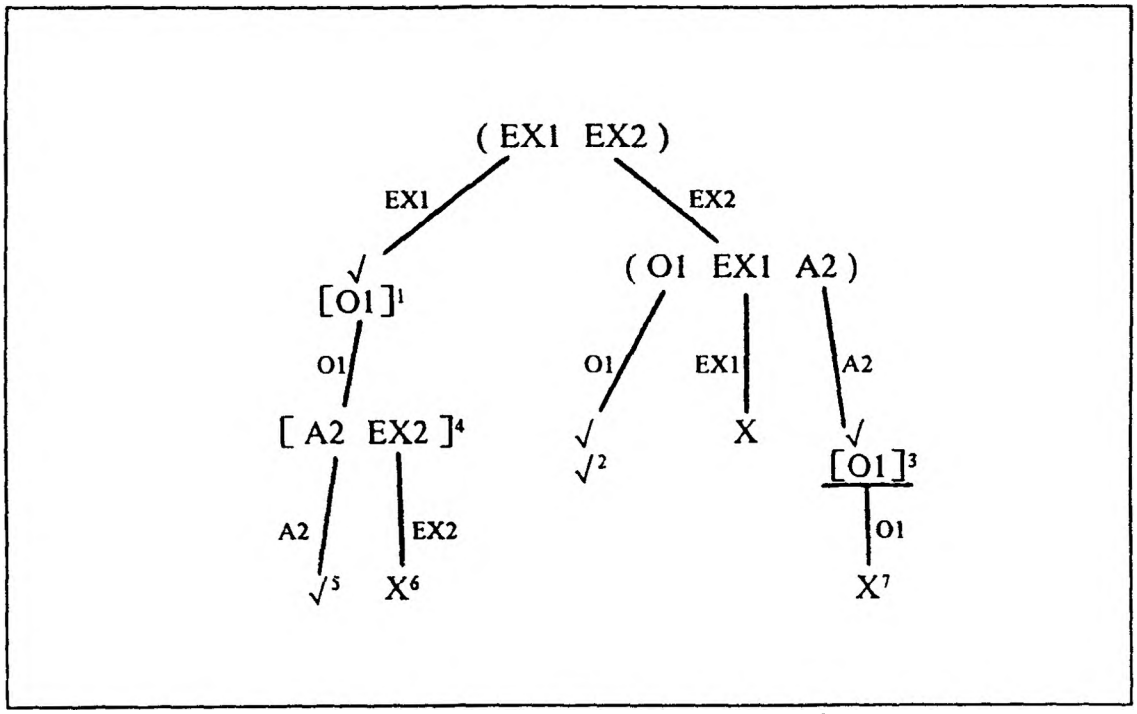

Figure 25. HS-tree for full adder after measurement information

A question arises concerning the amount of storage necessary to save all of the values and antecedants for nodes in the HS-tree which are labeled √. Clearly, the space requirement is dependent on the number of components and the number of diagnoses. However, the number of diagnoses which are possible for a system is also dependent on the number of components in the system. If a hierarchical approach is taken, the number of components involved at a level in the hierarchy is kept small compared to the total number of components at the lowest level system description.

D. MEASUREMENT HEURISTIC

The diagnostic problem is known to be underconstrained. Without sets of observations covering a range of behaviors for the device and a complete set of measurements for all internal components, there can be no guarantee of determining a single diagnosis which is, in fact, the actual diagnosis. Clearly, such exhaustive observations and measurements cannot be attained.

Diagnosis is generally an incremental process. When there are competing diagnoses for a system under a set of observations, measurements are taken or additional observations made in an attempt to reduce the number of possible diagnoses. However, not all measurements or additional observations are equally effective in reducing the size of the diagnosis set. Further, it may not be possible or cost effective to take certain measurements. A theory of measurements is needed to identify the best course to follow in order to reduce the number of diagnoses.

Reiter's theorem on measurements provides a very broad characterization of the effect of a measurement on the diagnosis set. Recall that the theorem states that every diagnosis which predicts some behavior Π (for example, a specific value of an output of some component) remains a diagnosis when Π is added to the observations as a measurement. Diagnoses which predicted $\neg\Pi$ are rejected. However, new diagnoses can arise when Π is added as a measurement. Any new diagnosis is a strict superset of some original diagnosis which predicted the behavior $\neg\Pi$. Therefore, when a measurement is added to the observations, the number of diagnoses can decrease or increase. Also, the individual diagnoses can become more complex, as they can involve more components.

Reiter's theorem provides no guidelines for where to take measurements. The problem lies with the possibility that an individual diagnosis can become more complex after a measurement is added to the observations. An important research

question is to characterize when a measurement *filters,* that is, only confirms or rejects additional diagnoses without adding any new diagnoses.

Consider the two bit adder-subtracter shown in Figure 31 under the set of observations given below.

(IN1 EX1) = 0
(IN2 EX1) = 0
(IN2 EX2) = 1
(IN2 EX4) = 1
(IN2 EX5) = 1
(OUT1 EX3) = 0
(OUT1 EX6) = 1
(OUT1 O2) = 1

Without additional information from other sets of observations or measurements, there are 17 diagnoses which explain the system behavior. These diagnoses are: (EX1), (EX2 EX5), (EX2 EX6), (EX2 A1), (EX2 A2), (EX2 O1), (EX3 EX5), (EX3 EX6), (EX3 A1), (EX3 A2), (EX3 O1), (EX2 EX4 O2), (EX2 EX4 A4), (EX2 EX4 A3), (EX3 EX4 O2), (EX3 EX4 A4), and (EX3 EX4 A3). Of the 17 possible diagnoses, only the single fault diagnosis (EX1) predicts that the output of component EX1 is 1. If the output of component EX1 is measured, regardless of the value found, the measurement will filter the set of diagnoses. The reason that the measurement filters the set is related to the fact that (EX1) is a single fault diagnosis and is not related to the number of diagnoses which predict a particular value for the output of component EX1. Only two diagnoses predict the output of component A4 to be 1. However, if the output of component A4 is measured and found to be 1, 27 diagnoses will result. Of these 27 diagnoses, only the two which predicted the output of component A4 to be 1 are from the original set. The other 25 diagnoses are supersets of some of the original 15 diagnoses which predicted the output of component A4 to be 0.

There is also a problem in characterizing what constitutes the "best" measurement to take. A measurement can be evaluated according to: 1) the potential for a large decrease in the number of diagnoses, 2) a significant decrease in the size of the diagnosis set regardless of the value found when the measurement is taken, 3) the complexity of the resulting diagnoses. Generally, no single measurement will perform best when evaluated with respect to each of these three criteria. However, it is usually the case that when a measurement produces a relatively small number of diagnoses, the individual diagnoses in the set are less complex.

In the adder-subtracter example just discussed, measuring the output of component EX1 offers the potential for the greatest decrease in the size of the diagnosis set. The effect of measuring the output of component EX2 is somewhat different. If the output of component EX2 is measured and found to be 0, nine diagnoses result. Nine diagnoses also result if the output of component EX2 is found to be 1. Thus, a measurement on component EX2 is an example of a measurement which yields a significant decrease in the size of the diagnosis set regardless of the result of the measurement. It should be noted that such a measurement does not always exist.

Path sensitization algorithms such as the *d*-algorithm are well known for testing devices. The *d*-algorithm is designed to determine settings for the inputs of a circuit which will result in a particular component being exercised. The algorithm does not select the component to be exercised. Rather, given a particular component and type of fault (specifically, a *stuck-at*), it specifies how to set the inputs of the device in order to test the component. The algorithm is part of a theory of testing rather than a theory of measurement.

An application area related to diagnostic testing is the generation of tests to be used in the quality control phase of the manufacturing process. The goal is to design a series of tests such that if a device passes all of the tests, one can be reasonably confident that the device is manufactured correctly. Often, these tests are also applied in a diagnostic setting. Useful information can be gained since the tests generally exercise individual components. Thus, given a component to be tested, the input settings used to test the component when it was manufactured can be used to test the component for diagnosis.

The problem of test generation is a difficult one. The *d*-algorithm is not general enough. It can only be applied at the gate level and gate level descriptions of complex devices are far too large for the algorithm to be effective. Singh [*Si*87] has developed a representation for a device which facilitates the test generation process. His method is more general than the *d*-algorithm in that devices need not be Boolean and a hierarchical approach is used. Work done by Shirley [*Sh*86] may prove useful for diagnostic testing within the framework of diagnosis from first principles. He has developed a means for generating tests for quality control which exploit the design information and the desired behavior of the device. Such a method for generating tests could become a significant part of an integrated, automated system for the design, manufacture, testing, and diagnosis of complex devices. However, as with any test generation procedure designed to be used in quality control and not diagnostic testing, the method does not address the problem of selecting the component to be tested.

Shirley and Davis [*SD*83] have addressed the problem of generating diagnostic tests for a device. The method is generally based on path sensitization and the goal is to "generate a test whose result depends only on parts known to be good and exactly one of the possibly faulty parts" ([*SD*83], page 455). The major weakness of their method is that it assumes a single point of failure and that failure is non-intermittent.

When the single fault assumption is applied, the choice of the component for which to generate a test is straightforward. One simply selects one of the components which represents a single fault diagnosis. The choice of the component to test becomes more difficult, and at the same time more crucial, when faced with a large number of multiple fault diagnoses for a device.

When the probability of failure of the individual components is available, that information can be of use in the determination of the components to be measured. The components which correspond to the most likely diagnosis are either measured directly or are indirectly tested through methods such as the *d*-algorithm or testing procedures such as that developed by Singh and Shirley and Davis. Another approach to the problem of determining what to measure is applied within expert systems. These systems often utilize experiential knowledge to suggest what to measure or test and how to carry out the test.

The difficulty that arises in relying on experiential knowledge and probability information is that such knowledge and information is not always available. This is generally the case in the domain which diagnosis from first principles was developed to address. One of the primary strengths of diagnosis from first principles is its applicability to the diagnosis of new, complex devices. Diagnosis from first principles allows automated diagnosticians for these devices to be developed much earlier than would be possible if an expert systems approach were used. When a first principles diagnostician is used, it is not necessary to wait for human expertise to be developed and then written in the form of rules.

A heuristic for suggesting what measurement to make or component to test has been developed. It does not require the use of any information which is not already included in the system description and observation set. Thus, it can be applied when

experiential knowledge and knowledge of the probability of component failures is not available.

When an input of some component *n* is connected to the output of some other component *m* either directly or indirectly, then *n* is dependent on *m*. Since this dependency is defined only through the physical interconnections of the components, this type of dependency will be referred to as a *structural dependency*. If the diagnoses for the device indicate that component *n* might be faulty, then measuring or testing any component on which component *n* is structurally dependent, including component *n* itself, may reduce the number of diagnoses.

Structural dependency, while of some value in selecting a component to be measured or tested, does not provide as strong of a criterion as is needed. Structural dependency does not take into account the specific information available concerning the observations under which the diagnoses were computed. *Functional dependency* combines the information contained in the structural dependencies with the observation data to provide the basis of a heuristic for selecting a measurement which will be valuable in reducing the number of diagnoses.

Component *n* is functionally dependent on component *m* if and only if *n* is structurally dependent on *m* and the value of the output of the component *m* must be known in order to determine the value of the output of component *n*. In addition, component *n* is functionally dependent on itself. Let the *dependency set* of a component *n* be {*m* | component *n* is functionally dependent on component *m*}.

Functional dependency is determined by structure as well as the value of the observed inputs for the device. The operation of the device is simulated and the antecedants of the predicted values are recorded in order to determine the dependency set for a particular component. A crucial aspect of the definition of functional

dependency is the value of the output of component *m* must be known in order to determine the output of component *n*.

Consider a device which has a component A_i where A_i is a logical *and* gate. Suppose that, based on the inputs to the circuit and the simulation of the device, the first input of A_i is predicted to be 0 and the second input is predicted to be 1. The output of A_i is functionally dependent on A_i and all components which determined the value of the first input to be 0. Those components which determined the value of the second input of A_i are not included in the dependency set unless, of course, they also contributed to the determination of the value of the first input of A_i.

Note that there may be more than one dependency set for an output of a component. For example, if both inputs to component A_i had been predicted to be 0, there would have been two choices for the dependency set: A_i and all components which led to the determination of the first input of A_i, or A_i and all components which led to the determination of the second input of A_i. When there are competing dependency sets, the set with the smallest number of components is selected. In the event of a tie, the selection of the dependency set may be made arbitrarily. This does not occur very often, but a question for future research is to determine whether there are criteria for breaking ties which improve the effectiveness of the heuristic.

The heuristic is based on the functional dependencies of the components and the minimal conflict sets which determine the diagnoses. It has the advantage of being relatively simple to implement and not requiring any information other than the input values from the observation set and the system description. Constraint propagation can be used to simulate the operation of the device and determine the dependency set for the components. The system description provides the rules by which the device is designed to operate and the inputs to the device provide the starting values. Thus, elements necessary to implement the heuristic are already part of the diagnostician.

The only requirement which may be lacking is that the dependency set should be minimal. The inference mechanism of a first principles diagnostician based on Reiter's theory of diagnosis need not produce a minimal set of antecedants for a value since minimal conflict sets are not required. It is, however, easier to determine minimal dependency sets than minimal conflict sets since dependency sets are produced through simulation while conflict sets must be computed through inferences. It should be noted that the heuristic can be applied using non-minimal dependency sets, but it is not as effective.

Assume that a single observation set is available for a device. First, the dependency set for each component under the given observations is determined. A weight which is based on the dependency sets and minimal conflict sets is then determined for each component. The component with the largest weight is selected to be measured or tested. The procedure for calculating the weights is shown in Figure 26.

Suppose that the following input and output values have been observed for the two bit adder-subtracter shown in Figure 31.

IN1(EX1) = 0
IN2(EX1) = 0
IN2(EX2) = 0
IN2(EX4) = 1
IN2(EX5) = 1
OUT1(EX3) = 0
OUT1(EX6) = 1
OUT1(O2) = 0

The symptoms of the faulty component(s) are that the outputs of components EX6 and O2 are incorrect. The above observation set leads to the diagnoses: (EX4), (O2 A1), (O2 A2), (O2 O1), (O2 EX5), (O2 EX6), (A3 A1), (A3 A2), (A3 O1), (A3 EX5), and (A3, EX6).

Procedure **Determine-Weight** (COMPONENTS, OBS, S)

Let S be the collection of minimal conflict sets for the observations.

For each $c \in$ COMPONENTS, $W_c = 0$

For each $c \in$ COMPONENTS determine the dependency set for c for the inputs.

```
For each mcs ∈ S
   For each c ∈ mcs
      For each component d such that c is functionally dependent on d
         W_d = W_d + 1
      End For
   End For
End For
```

Figure 26. Procedure for calculating the weight of the components

Component EX4 is identified by the heuristic for selecting the component to be tested. Any component which is a single fault diagnosis appears in every minimal conflict set and is therefore emphasized by the weighting function. This is appropriate, but does not illustrate the strength of the heuristic. Single fault diagnoses are considered the most likely diagnoses. Certainly, when there are eleven diagnoses, with one being a single fault diagnosis and ten being multiple fault diagnoses, the obvious choice for the component to be measured is the component involved in the single fault diagnosis.

Suppose that the output of component EX4 is measured and found to be 1. As this is the value that is predicted for the output when EX4 is functioning correctly, the single fault diagnosis involving EX4 is eliminated by this measurement. The measurement does not produce any new diagnoses, so the diagnosis set now contains

the ten double fault diagnoses from the original set. The minimal conflict sets which produce these diagnoses are {O2 A2} and {EX5 EX6 A1 A2 O1}.

With the measurement OUT1(EX4)= 1 included, the dependency set for each component is:

COMPONENT	DEPENDENCY SET
EX1	{EX1}
EX2	{EX1 EX2}
EX3	{EX1 EX2 EX3}
EX4	{ }
EX5	{EX5}
EX6	{EX5 EX6 A1 A2 O1}
A1	{A1}
A2	{A2}
A3	{A3}
A4	{EX5 A4}
O1	{A1 A2 O1}
O2	{A2 O2}

Since the value for the output of EX4 is given, EX4 does not appear in any dependency set. More than one dependency set exist for components A1, A2, and A4. These components are logical *and* gates with both inputs equal to 0. The dependency set shown above for each of these components represents the set with the fewer number of components. The weight of each component is calculated according the the procedure in Figure 26. The resulting weights, W, are:

$$W_{EX1} = 0$$
$$W_{EX2} = 0$$
$$W_{EX3} = 0$$
$$W_{EX4} = 0$$
$$W_{EX5} = 2$$
$$W_{EX6} = 1$$
$$W_{A1} = 3$$
$$W_{A2} = 3$$
$$W_{A3} = 2$$
$$W_{A4} = 0$$
$$W_{O1} = 2$$
$$W_{O2} = 1$$

The component with the largest weight is selected to be measured. In this case, there is a tie between components A1 and A2. Measuring the output of either of these components, in fact, offers the same potential for reducing the number of

diagnoses. If the output of component A1 is measured and found to be 0, eight diagnoses result. If it is found to be 1, two diagnoses result. The same is true if the output of component A2 is measured.

Table IV shows the results of all possible single measurements under the given inputs and outputs with the previous measurement on component EX4 also included. A measurement on component EX5, A1 or A2 offers the potential for the greatest reduction in the number of diagnoses, followed by (in order) components O1, A3, and A4. Note that this ordering corresponds very well to the ordering of the weights which were determined by the heuristic.

Table IV. EFFECT OF MEASUREMENTS ON THE NUMBER OF DIAGNOSES

COMPONENT	No. of Diagnoses If Output = 0	No. of Diagnoses If Output = 1	Rank by Weight
EX1	10	20	4
EX2	10	20	4
EX5	8	2	2
A1	8	2	1
A2	8	2	1
A3	5	5	2
A4	10	8	4
O1	4	6	2

Several tests were made in order to evaluate the effectiveness of the heuristic for selecting the component to be measured. Three combinations of input and output values were randomly selected for each device from combinations which result in several diagnoses. In addition, the combinations were selected so that there would be a choice of several components whose output could be measured. The heuristic was applied and the effect of measuring the output of the component was evaluated with respect to reducing the number of diagnoses. The effect of measuring the output of the other components which appear in a diagnosis was also determined.

Table V summarizes the results of these tests. The circuits used in the tests appear in the Section C of Chapter VI. The third column shows the number of diagnoses without measurement information. The fourth column is the number of diagnoses which can result if the component selected by the heuristic is measured. The fifth column is the rank of the selected component among all components which could be measured, with respect to the potential decrease in the size of the diagnosis set. The last column is the number of components which were involved in diagnoses and could be measured.

Table V. EVALUATION OF HEURISTIC FOR SELECTING COMPONENT

Circuit	Test Number	Original Diagnoses	Diag. After Measurement	Rank of Measurement	Possible Measurements
Full adder	1	4	1	1	3
	2	4	1	1	3
	3	4	2	1	3
Two bit Adder	1	10	4	2	4
	2	4	3	3	4
	3	7	1	1	4
Overflow detector	1	13	1	1	14
	2	11	1	1	10
	3	31	7	6	11
Adder-subtracter	1	12	1	1	9
	2	28	13	3	9
	3	53	22	5	9
BCD to binary	1	21	3	1	6
	2	22	8	3	8
	3	12	3	1	5
Decoder	1	9	3	1	6
	2	94	30	4	13
	3	21	9	4	9

VI. IMPLEMENTATION OF THE DIAGNOSTICIAN

A. INTRODUCTION

A significant part of this work has been the implementation of a diagnostician based on Reiter's theory of diagnosis. Many diagnostic problems have been studied using this implementation and this led to the discovery of the error related to pruning the HS-tree. Having an implemented diagnostician provided a means for the development and testing of the measurement heuristic. In this chapter, an overview of the implementation is presented. Section C presents several devices which have been diagnosed by the system.

B. IMPLEMENTATION NOTES

The diagnostician is written in Common Lisp and has been developed on a MicroVAX II running VMS. All of the CPU times which are reported are from runs on the MicroVAX. All components, component types, component functions, constants, variables, elements of HS-tree node labels and HS-tree edge labels are represented by LISP atoms. Many of these atoms have information stored on their property lists, as discussed in the following sections. Throughout the description of the implementation, the representation and diagnosis of the full adder will be used as an example.

There are two major groups of functions within the diagnostician: those functions related to building the HS-tree and those functions which implement the inference mechanism. This follows Reiter's theory of diagnosis from first principles and allows questions related to the HS-tree and questions related to the inference mechanism to be treated separately.

1. Implementation of the HS-tree.

The HS-tree consists of two structures: a structure which contains the node label information from the tree and a structure which contains the edge label information. Both of these structures are lists.

In the node structure, the first element of the list is the root level of the HS-tree, the second element is level 1 of the HS-tree and so on. Within a level, the elements appear in the left-to-right order in which they occur in the HS-tree. The tree construction algorithm can determine when the tree is complete by examining the list which is the last element of the node structure. If every element of this list is either *T* (represents √) or *nil* (represents ×), the tree is complete.

In processing the HS-tree, many computations involve the collection of edge labels on the path from the root to a node. These sets are used in subset tests to determine if a node can be closed and ultimately represent the diagnoses for the device. The edge labels could be computed from the information in the node label structure, but it is more efficient to maintain the edge labels in a separate structure. In the path structure, the *n*th element is a list of the collections of edge labels from the root to the nodes at level $n-1$ in the tree. The first element of the path structure is (*nil*) since the root is at the root level. For every node value in the node structure, the edge labels on the path to that node are found in the corresponding location in the path structure.

For convenience, Figure 27 repeats the HS-tree for the full adder which has appeared elsewhere in this dissertation. The representation of the tree is:

NODE structure: (((EX1 EX2)) (*T* (O1 EX1 A2)) (*T nil T*))

PATH structure: ((*nil*) ((EX1) (EX2)) ((EX2 EX1) (EX2 A2) (EX2 O1)))

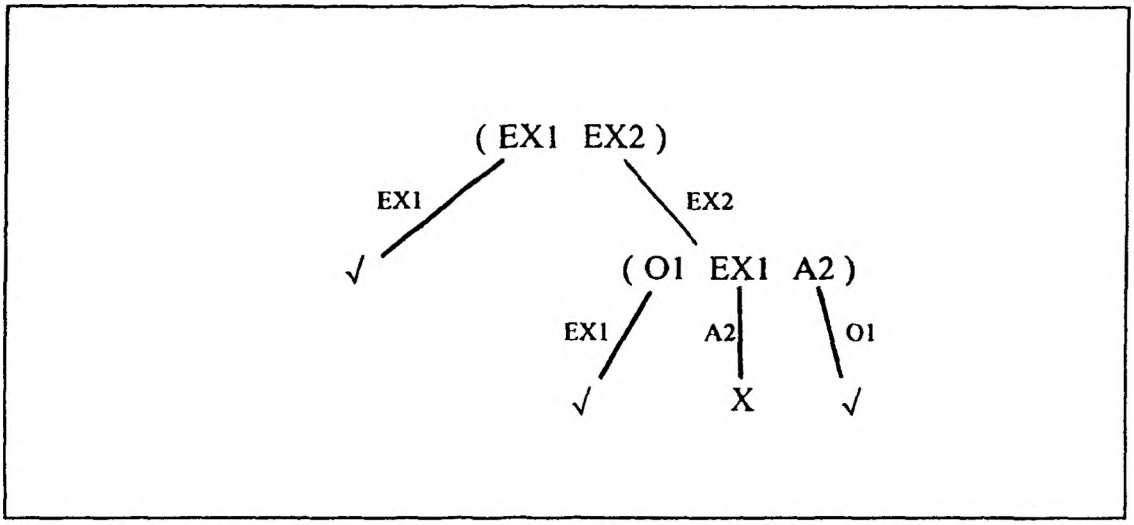

Figure 27. HS-tree for full adder

The construction of the HS-tree follows Reiter's algorithm except that pruning by the removal of redundant edges is not used. Also, when there is a choice of labels to be reused, the set with the smallest number of elements is used. This does not affect the number of calls to the underlying theorem prover, but it is a way of keeping the tree smaller. By using the smallest possible label for a node, if both a minimal conflict set and a superset of the minimal conflict set could be reused, the minimal conflict set is used as the label.

2. Implementation of the inference mechanism.

The representation of a device is basic to the operation of the inference mechanism. A global variable, *COMPONENTS*, is a list of the names of all components in the device. Each component is an atom and has the following properties associated with it.

ABnormal: T or nil

type: EXORG, ANDG, ORG, etc.

numinputs: Integer representing the number of input lines of the component. The input lines are referred to as IN1, IN2, ...

numoutputs: Integer representing the number of output lines of the component. The output lines are referred to as OUT1, OUT2, ...

line values: All of the input and output values for a component are stored on the property list of the component under the input or output name.

The value for a particular input or output is an ordered list of value information. Each element of the list contains the value associated with the input or output at a particular time or state. The value information for a state consists of the value itself and the list of antecedant components which determined the value. State numbers can be any non-negative integer, although the value information is stored in order starting with the information for state 0.

The variable *Connection-List*, also global, defines the connections. Each element of *Connection-List* is a pair which describes a connection between two components, for example, ((OUT1 EX1) (IN1 EX2)). The order of the elements in the list, both within the pair and within *Connection-List*, is not significant.

Each output of a component type has associated with it the pattern which defines the function that the output represents and the inputs to which the function is applied. This is stored on the property list of the component type under the output name. For example, OUT1 of EXORG would have the pattern (EXOR IN1 IN2) associated with it. The definition of function is stored under the property *definition* on the function name. Definitions must be provided for all functions of the components. The information which would be provided for the full adder is shown in Figure 28.

The information concerning the components, connections, output patterns, function definitions, and observed values is defined once for the device to be diagnosed. When the inference mechanism is invoked, the values associated with the

components' inputs and outputs are initialized to null values. The inference mechanism then uses the observed values along with the data stored on the property lists to determine consistency or compute the conflict set.

```
COMPONENTS = (EX1 EX2 A1 A2 O1)

Connection-List = (((OUT1 EX1) (IN1 EX2))
                   ((OUT1 EX1) (IN2 A2))
                   ((OUT1 A2) (IN1 O1))
                   ((OUT1 A1) (IN2 O1))
                   ((IN1 EX1) (IN1 A1))
                   ((IN2 EX1) (IN2 A1))
                   ((IN1 A2) (IN2 EX2)))

type on EX1 = EXORG
type on EX2 = EXORG
type on A1 = ANDG
type on A2 = ANDG
type on O1 = ORG

numinputs on EX1 = 2
numinputs on EX2 = 2
numinputs on A1 = 2
numinputs on A2 = 2
numinputs on O1 = 2

numoutputs on EX1 = 1
numoutputs on EX2 = 1
numoutputs on A1 = 1
numoutputs on A2 = 1
numoutputs on O1 = 1

OUT1 on EXORG = (EXOR IN1 IN2)
OUT1 on ANDG = (AND IN1 IN2)
OUT1 on ORG = (OR IN1 IN2)
```

Figure 28. Representation of the full adder used in the implementation

When the inference mechanism is invoked to determine the label for a node, n, the following operations precede the call. First, the property *ABnormal* is set to T for all components in $H(n)$ and it is set to *nil* for all members of

COMPONENTS – $H(n)$. Second, elements which define each output of every abnormal component to be a unique variable are added to the observed values. The antecedant list of such a variable value is empty.

The inference mechanism is invoked by calling the function Process-Values and passing it the observed values. The list of values is processed one element at a time. When a value from the list is processed, it is compared to the value already stored on the component for the particular input or output. If an inconsistency is found, the inference mechanism immediately returns the conflict set, which is the union of the antecedants of the inconsistent values. If no inconsistency is found, the value being processed is stored on the property list as long as it provides new information. A value provides new information if it is more specific. For example, if the value already stored is a variable and the value being processed is a constant, the stored value is updated to the constant value. If the stored value is a constant, it is not replaced by a variable.

When a new value is recorded, that value is propagated through the device in two ways. Based on the information in the *Connection-List*, the value is passed along the defined connections. These new values are added to the list of the values to be processed. The other way that a value can be propagated is through a component. If a component is not abnormal, values may be propagated by applying the definition of the function of the component. However, values cannot be propagated through a malfunctioning component.

The values are not processed in any particular order and, as a result, the conflict sets returned by the inference mechanism are not necessarily minimal. However, the conflict sets are kept close to minimal. When the value being processed agrees with the already recorded value, the two antecedant lists may not be the same as a value

can often be determined in more than one way. When this occurs, the antecedant list with the fewer components is retained.

C. EXAMPLES

The following circuits have been referred to throughout the text. For each example, a diagram of the circuit and the diagnoses for a given set of observations are presented. In the diagrams of the circuits, standard logic symbols are used and the inputs and outputs of a component are numbered consecutively from the top.

1. Two Bit Adder.

The circuit shown in Figure 29 adds two two-bit binary numbers in parallel. IN1 of EX1 is the least significant bit of the first number and IN1 of EX2 is the most significant bit of the first number. The least significant bit of the second number is IN2 of EX1 and the most significant bit is IN2 of EX2. The output of EX1 is the least significant bit of the sum, the output of EX3 is the most significant bit of the sum and the output of O1 is the carry out. Given the input and output values shown below, the diagnoses are: (A1), (EX2), (EX3 O1), and (EX3 A3). Determining the diagnoses required 4.90 seconds of CPU time.

IN1(EX1) = 1
IN2(EX1) = 1
IN1(EX2) = 1
IN2(EX2) = 0
OUT1(EX1) = 0
OUT1(O1) = 0
OUT1(EX3) = 1

2. Overflow Detector.

The following circuit is taken from ([*HP*81], page 39) and is an additive overflow detector. The meaning of the inputs and output of the circuit is:

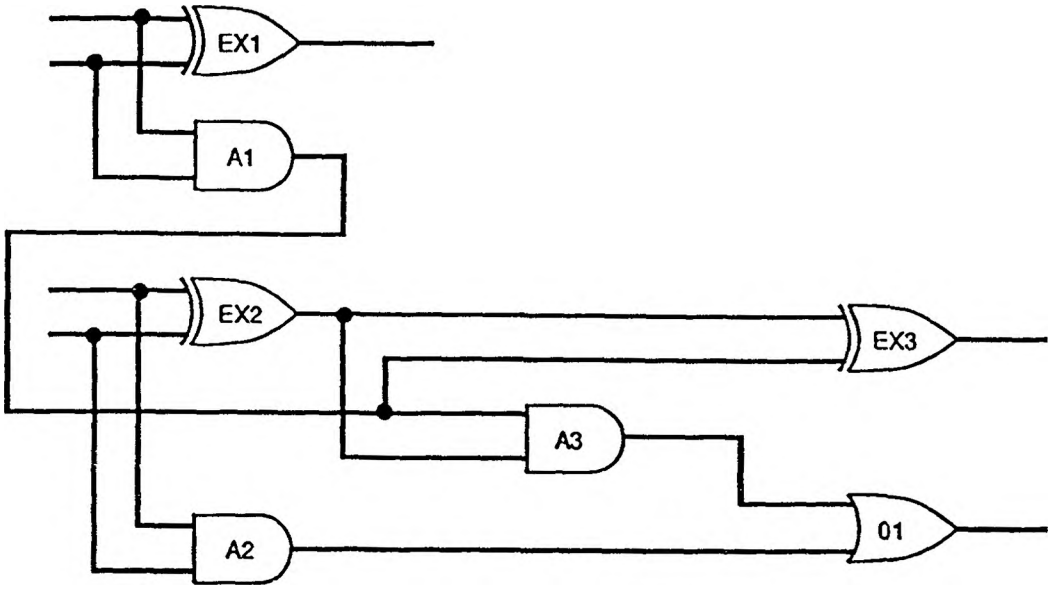

Figure 29. Diagram for the two bit adder

Operation is addition	IN1 of A1 = 1
Operation is subtraction	IN1 of A5 = 1
Sign of the augend is negative	IN1 of N1 = 1
Sign of the addend is negative	IN2 of A4 = 1
Sign of the minuend is negative	IN1 of N4 = 1
Sign of the subtrahend is negative	IN2 of A6 = 1
Carry from the most significant digit	IN2 of A1
Additive overflow error has occurred	OUT1 of O3

The diagnoses for the overflow detector under the following inputs and output are: (A12), (A11), (O2), (O1), (A9), (A10), (O3), (A7, A8), (A3 A4), (A5 A6), (A1 A2), and (A1 N1). Determining the diagnoses required 85.46 seconds of CPU time.

IN1(A1) = 0
IN2(A1) = 0
IN1(N1) = 1
IN1(N2) = 0
IN1(A5) = 0
IN1(N4) = 0
IN2(A6) = 1
OUT1(O3) = 1

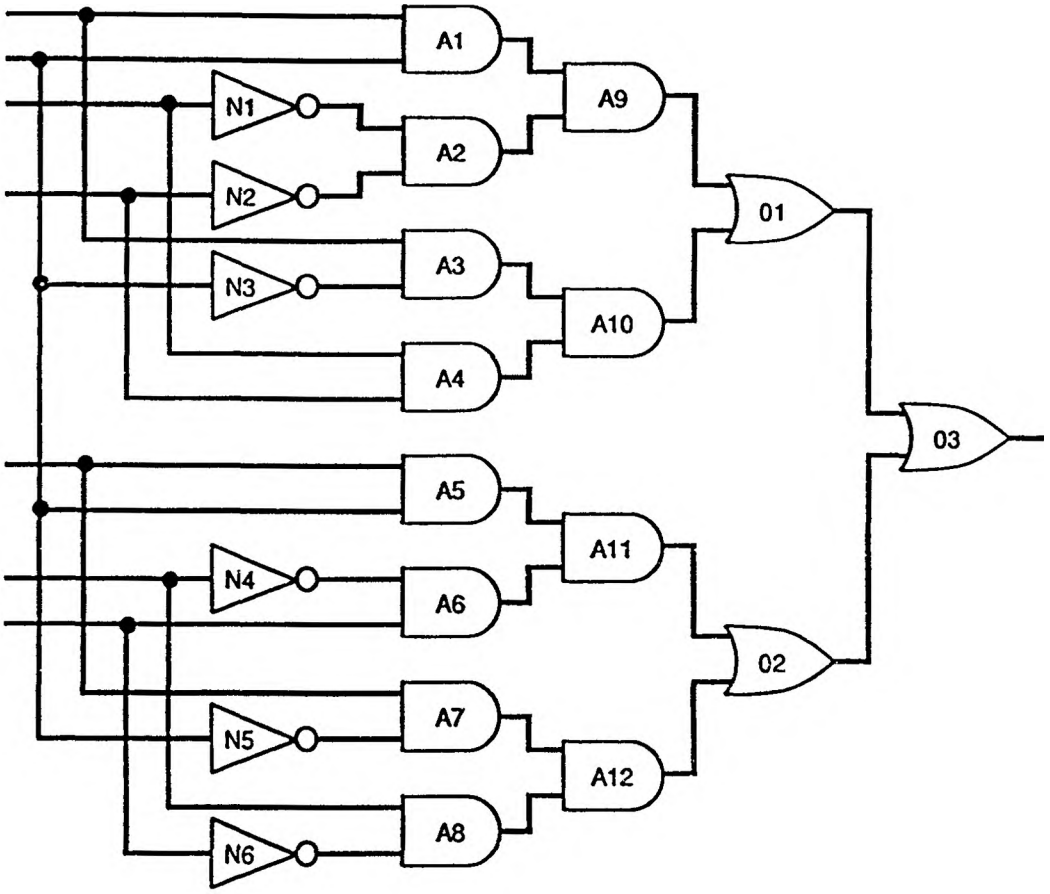

Figure 30. Diagram for an overflow detector

3. Two Bit Adder-subtracter.

The circuit in Figure 31 performs addition or subtraction on two two-bit numbers. IN1 of EX1 is the control bit. If the control bit is 0, the circuit performs the addition of the binary numbers. If the control bit is 1, subtraction is performed. The least significant bit of the sum/difference is OUT1 of EX1, the most significant bit is OUT1 of EX6 and OUT1 of O2 is the carry out.

The diagnoses for the 2 bit adder-subtracter under the following inputs and outputs are: (EX1), (EX2 EX5), (EX2 EX6), (EX2 A1), (EX2 A2), (EX2 O1), (EX3 EX5), (EX3 EX6), (EX3 A1), (EX3 A2), (EX3 O1), (EX2 EX4 O2), (EX2 EX4 A4), (EX2 EX4 A3), (EX3 EX4 O2), (EX3 EX4 A4), and (EX3 EX4 A3). Determining the diagnoses required 34.43 seconds of CPU time.

IN1(EX1) = 0
IN2(EX1) = 0
IN2(EX2) = 1
IN2(EX4) = 1
IN2(EX5) = 1
OUT1(EX3) = 0
OUT1(EX6) = 1
OUT1(O2) = 1

4. BCD to Binary Converter.

The circuit in Figure 32 ([*JK*87], page 160) converts a decimal digit stored in a binary coded decimal (BCD) 4 bit code to its binary equivalent. IN1 of component N1 is the least significant bit and IN1 of component N4 is the most significant bit of the BCD value. OUT1 of EX1 is the least significant bit and OUT1 of O3 is the most significant bit of the binary value.

The diagnoses for the BCD to binary converter under the following inputs and outputs are: (EX1 O2), (EX1 N3), (EX1 N1), and (EX1 A4). Determining the diagnoses required 13.22 seconds of CPU time.

IN1(N1) = 0
IN1(N2) = 0
IN1(N3) = 0
IN1(N4) = 1
OUT1(EX1) = 0
OUT1(O1) = 0
OUT1(O2) = 0
OUT1(O3) = 0

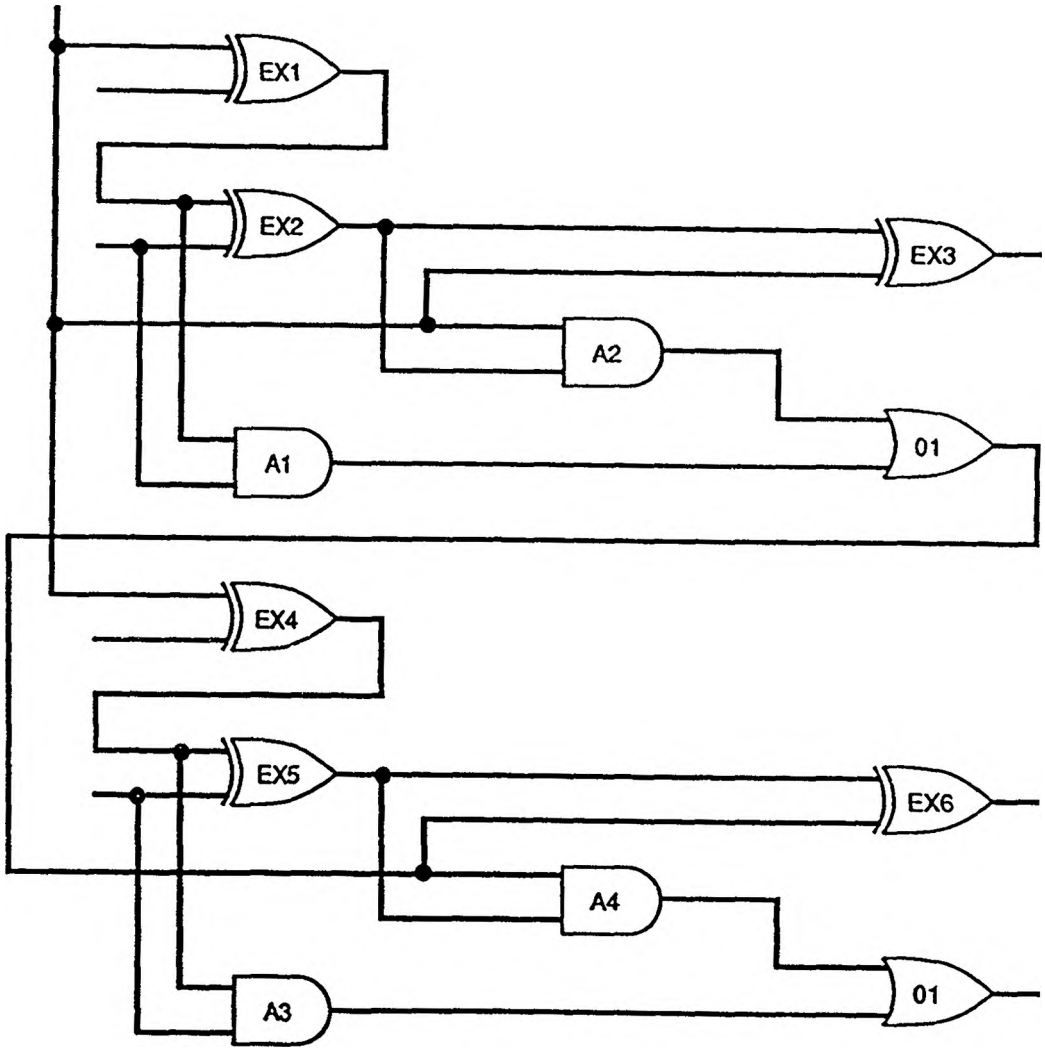

Figure 31. Diagram for the 2 bit adder-subtracter

5. Decoder.

An n-to-2^n line decoder is a combinational circuit for which all of the output values are 1 except a single selected line whose output is 0. There is a one-to-one correspondence between the selected line and the value of the input. In the 4-to-16

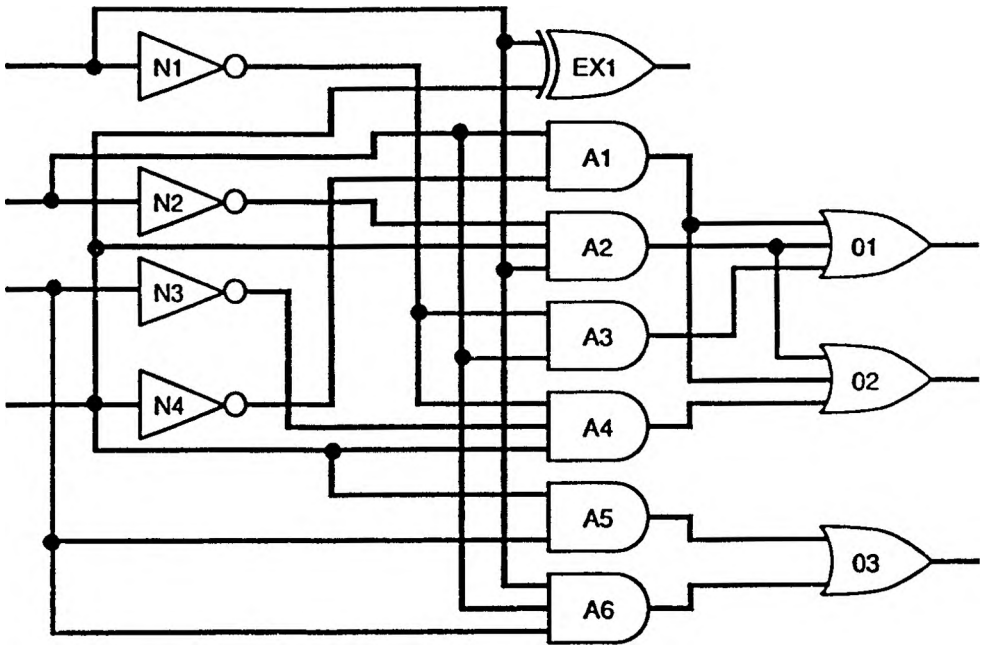

Figure 32. Diagram for a BCD to binary converter

line decoder shown in Figure 33 ([*JK*87], page 179), IN1 of component NA4 is the most significant bit and IN1 of NA1 is the least significant bit of the 4 bit input. OUT1 of component NA5 is the selected line for inputs of 0000 and OUT1 of NA20 is the selected line for inputs of 1111.

The diagnoses for the 4-to-16 line decoder under the following inputs and outputs are: (NA4), (NA5), (NA2), (NA1), (A1), (NA3), and (A5). Determining the diagnoses required 63.21 seconds of CPU time.

IN1(NA1) = 0
IN1(NA2) = 0
IN1(NA3) = 0
IN1(NA4) = 0
OUT1(NA5) = 1
OUT1(NA6) = 1
OUT1(NA7) = 1
OUT1(NA8) = 1
OUT1(NA9) = 1
OUT1(NA10) = 1
OUT1(NA11) = 1
OUT1(NA12) = 1
OUT1(NA13) = 1
OUT1(NA14) = 1
OUT1(NA15) = 1
OUT1(NA16) = 1
OUT1(NA17) = 1
OUT1(NA18) = 1
OUT1(NA19) = 1
OUT1(NA20) = 1

6. Sequential Circuit.

The circuit illustrated in Figure 34 is a simple sequential circuit. This example is taken from ([*JK*87], page 214). The feedback within the flipflop is not represented in the model. The flipflop functions simply as another component within the circuit. The inference mechanism assumes that the clock is functioning correctly and is thus unable to diagnose faults which arise as a result of timing problems. As such, the diagnostician should be viewed as diagnosing faults which occur as a result of the failure of one or more components.

Applying the assumption that the timing of the device is not at fault is analagous to the use of the assumption of ideal connections. When assumptions such as these are applied, there is the risk that the description of the device and inferences concerning its operation do not match those of the actual device. However, at this point in the development of diagnosis from first principles, these assumptions are necessary.

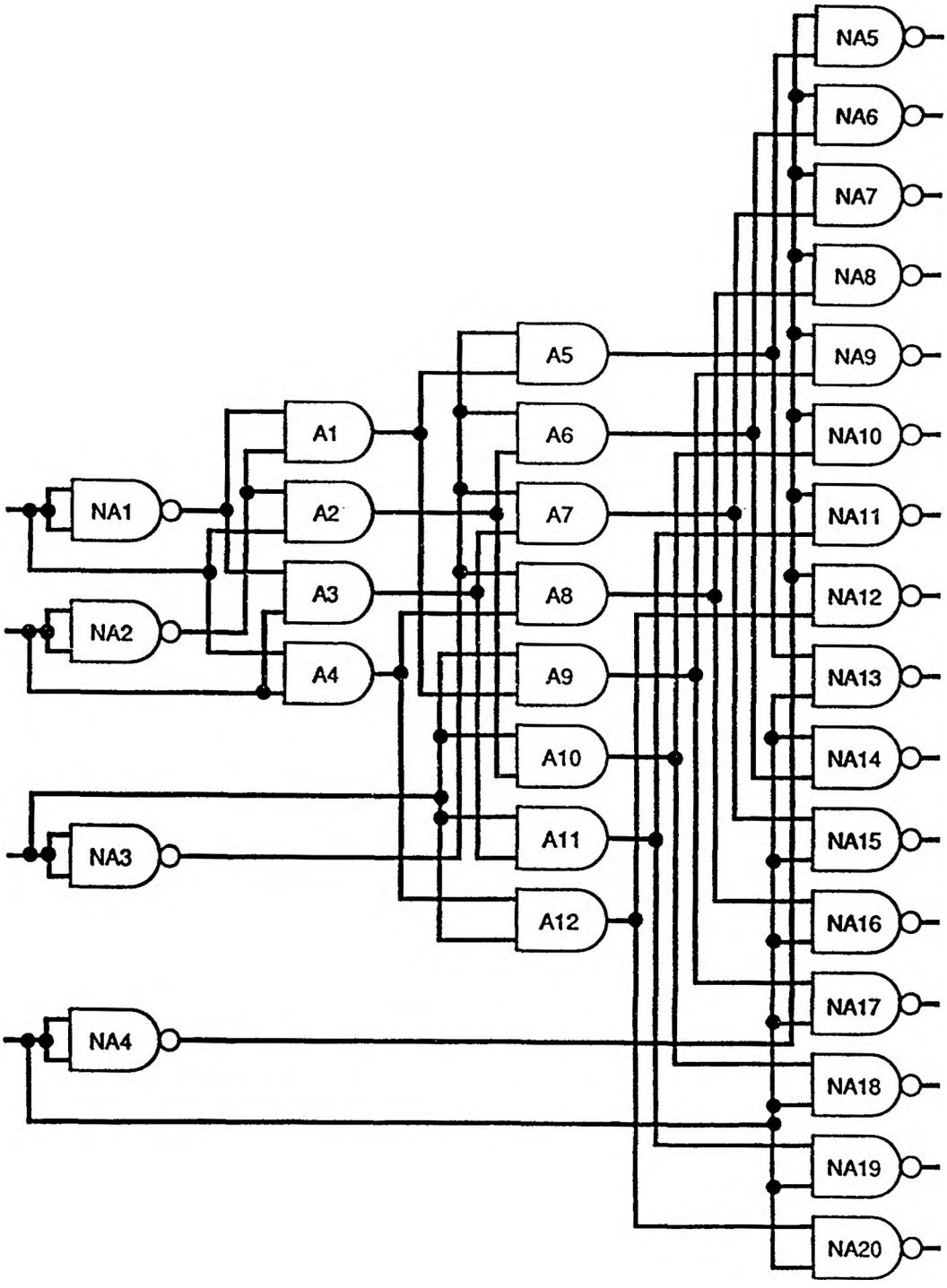

Figure 33. Diagram for a 4 to 16 line decoder

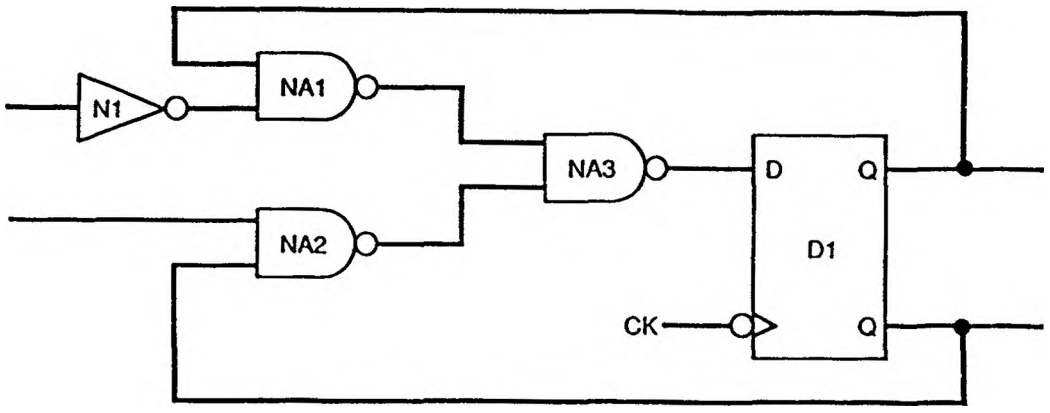

Figure 34. Diagram of a circuit utilizing a flipflop and feedback

7. Arithmetic Circuit.

The device in Figure 35 has been used in examples by Davis [*Da*84], de Kleer and Williams [*DW*87], and Reiter [*Re*87]. The inference mechanism of this diagnostician is not yet capable of handling the arithmetic reasoning which is necessary to diagnose the device. However, it is possible to utilize a a symbolic mathematics package such as MACSYMA as part of the inference mechanism so that devices such as this can be diagnosed. At this time, the link to MACSYMA is not automatic, so the user acts as the interface between the diagnostician and the mathematics package. Under the inputs and outputs shown below, the diagnoses are: (M1), (A1), (M2 M3), (A2 M2).

IN1(M1) = 3
IN2(M1) = 2
IN1(M2) = 3
IN2(M2) = 2
IN1(M3) = 3
IN2(M3) = 2
OUT1(A1) = 10
OUT1(A2) = 12

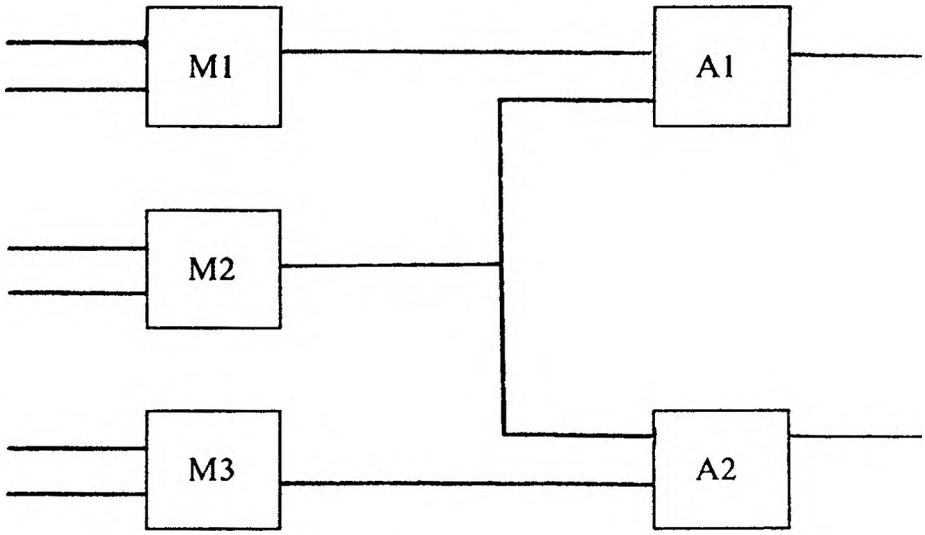

Figure 35. Diagram of an arithmetic circuit

VII. CONCLUSION AND FUTURE RESEARCH

Diagnosis from first principles is one of the most recent areas of application of Artificial Intelligence techniques. The foundation of diagnosis based on the structure of a device has been built in earlier work. This work has addressed a number of issues, both pragmatic and theoretical in nature, which are important to the overall development of first principles diagnosticians.

The community of researchers in Artificial Intelligence has long been divided into two camps: those who pursue the theoretical development of an area while ignoring questions of implementation and those who believe in building (or attempting to build) first and perhaps providing a theoretical foundation later. The premise of this work has been that theoretical development and implementation should parallel each other in Artificial Intelligence research.

The problem of suggesting measurements the result of which will decrease the number of diagnoses is an example of why theoretical and pragmatic issues should be addressed simultaneously. The theory of diagnosis, at its current level of development, provides no guidelines as to where to take measurements. It may be that the problem of determining the best measurement to take is, in general, unsolvable. Thus, heuristics such as the one presented in this work are important if effective diagnosticians are to be built.

There are many open questions concerning both the theory and implementation of diagnosticians based on the first principles approach. Two of the most significant questions which must be addressed are the removal of the assumption of ideal connections and the development of first principles diagnosticians which can handle devices with feedback. Some of the significant open questions are briefly discussed below.

In order to diagnose devices with feedback, suitable representations and inference mechanisms must be developed. Temporal logic has been used as the basis of the representation ([*Bo*82], [*Mo*85]) of devices with feedback in the same way that first order logic is used as the basis for representing combinational devices. In addition, temporal logic has been used by Browne et al. [*BC*86] in verifying sequential circuits. It is known that the problems of verification and diagnosis are related, so it may be that the reasoning used in the verification process can be extended for use in diagnosis.

Expert systems have been used very successfully for diagnosis. Typically, these systems can "explain" the reasoning behind a diagnosis by tracing the rules which were applied. Such explanations are a crucial part of the process of debugging the system and extending the rule set. The role of explanations in diagnosis from first principles is a question which has not been addressed.

A heuristic for selecting the component to be measured has been presented in this work. Other heuristics need to be developed and tested. However, a more significant research result would be the development of a method of determining, without invoking the inference mechanism, what effect a measurement is going to have on the minimal conflict sets. If such a method were available, it would be possible to determine the best component to measure. It would also allow the new diagnoses to be computed without calling the underlying theorem prover.

In [*RS*88], the use of heuristics and shortcuts within the inference mechanism is discussed. For example, it is often the case when diagnosing a malfunctioning device that a device of the same design which is working is available. If the device is complex, it may be more efficient to use the functioning device to determine values than to use the system description. Methods such as this are important because they

represent a way of dealing with the complexity of the reasoning task without weakening the power of the diagnostician.

An ultimate goal should be the development of diagnosticians which integrate the methods of expert systems and diagnosis from first principles. The integrated approach should be dynamic in that the diagnostician should learn from experience. This would allow for the most effective use of both methods of diagnosis.

REFERENCES

[*AI*84] "Special issue on qualitative reasoning about physical systems." *Artificial Intelligence* **24** (1984).

[*Bo*82] Bochmann, G. (1982) "Hardware specification with temporal logic: an example." *IEEE Transactions on Computers* **C-31** (3), pp. 223-231.

[*BC*86] Browne, M., Clarke, E., Dill, D. and Mishra, B. (1986) "Automatic verification of sequential circuits using temporal logic." *IEEE Transactions on Computers* **C-35** (12), pp. 1035-1043.

[*CL*73] Chang, C. and Lee, R. (1973) *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York.

[*CM*81] Clocksin, W. and Mellish, C. (1981) *Programming in Prolog*. Springer-Verlag, New York.

[*CP*87] Chandrasekhar, M., Privitera, J. and Conradt, K. (1987) "Application of term rewriting techniques to hardware design verification." *Proceedings of the 24th ACM/IEEE Design Automation Conference*, pp. 277-282.

[*Da*84] Davis, R. (1984) "Diagnostic reasoning based on structure and behavior." *Artificial Intelligence* **24** pp. 347-410.

[*dK*76] de Kleer, J. (1976) "Local methods for localizing faults in electronic circuits." **MIT AI Memo 394** Cambridge, MA.

[*DS*80] de Kleer, J. and Sussman, G. (1980) "Propagation of constraints applied to circuit synthesis." *Circuit Theory and Applications* **8** pp. 127-144.

[*DW*87] de Kleer, J. and Williams, B. (1987) "Diagnosing multiple faults." *Artificial Intelligence* **32** pp. 97-130.

[*Ge*82] Genesereth, M. (1982) "Diagnosis using hierarchical design models." *Proceedings of the National Conference on Artificial Intelligence*, Pittsburgh, PA, pp. 278-283.

[*Ge*84] Genesereth, M. (1984) "The use of design descriptions in automated diagnosis." *Artificial Intelligence* **24** pp. 411-436.

[*GS*88] Greiner, R., Smith, B. and Wilkerson, R. (1988) "A correction to the algorithm in Reiter's theory of diagnosis from first principles.", unpublished manuscript.

[*Ha*88] Hansen, T. (1988) "Diagnosing multiple faults using knowledge about malfunctioning behavior." *Proceedings of First International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pp. 29-36.

[*HP*81] Hill, F. and Peterson, G. (1981) *Introduction to Switching Theory and Logical Design*, John Wiley and Sons, New York.

[*Hs*85*a*] Hsiang, J. (1985) "Refutational theorem proving using term rewriting systems." *Artificial Intelligence* **25** pp. 255-300.

[*Hs*85*b*] Hsiang, J. (1985) "Two results in term rewriting theorem proving." *Proceedings of the Conference on Rewriting Techniques and Applications*, J. Jouannaud, ed., *Lecture Notes in Computer Science* **202** pp. 301-324.

[*JK*87] Johnson, E. and Karim, M. (1987) *Digital Design*, PWS Publishers, Boston.

[*KB*70] Knuth, D., and Bendix, P. (1970) "Simple word problems in universal algebras." *Computational Problems in Abstract Algebras*, J. Leech, ed., Pergamon Press, Oxford, England, pp. 263-297.

[*KH*88] Kalpin, S., Hadden, D. and Volovik, L. (1988) "A general architecture for factory based diagnosis of electronics." *Proceedings of First International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pp. 100-107.

[*LL*88] Lee, C. N., Liu, P., Clark, S. J. and Chiu, M. Y. (1988) "A hierarchical symptom classification for model based causal reasoning." *Proceedings of First International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pp. 54-59.

[*LP*81] Lewis, H. and Papadimitriou, C. (1981) *Elements of the Theory of Computation*, Prentice Hall, Englewood Cliffs, NJ.

[*Mo*85] Moszkowski, B. (1985) "A temporal logic for multilevel reasoning about hardware." *IEEE Computer* **18** (2), pp. 10-19.

[*OL*84] Overbeek, R. and Lusk, E. (1984) *The Automated Reasoning System ITP - User's Manual*, Technical Report ANL-84-27, Argonne National Laboratory.

[*PS*81] Peterson, G. and Stickel, M. (1981) "Complete sets of reductions for some equational theories." *Journal of the Association for Computing Machinery*, **28** pp. 233-264.

[*Re*87] Reiter, R. (1987) "A theory of diagnosis from first principles." *Artificial Intelligence*, **32** pp. 57-95.

[*Ro*65] Robinson, J. A. (1965) "A machine-oriented logic based on the resolution principle." *Journal of the Association for Computing Machinery*, **12** pp. 23-41.

[*RS*88] Reed, N. and Stuck, E. (1988) "Specialized strategies: an alternative to first principles in diagnostic problem solving." *Proceedings of National Conference on Artificial Intelligence*, St. Paul, MN, pp. 364-368.

[*SD*83] Shirley, M. and Davis, R. (1983) "Generating distinguishing tests based on hierarchical models and symptom information." *Proceedings of IEEE International Conference on Computer Design: VLSI in Computers*, pp. 455-458.

[*Sh*86] Shirley, M. (1986) "Generating tests by exploiting designed behavior." *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, pp. 884-890.

[*Si*87] Singh, N., *An Artificial Intelligence Approach to Test Generation*, Kluwer Academic Publishers, Norwell, MA.

[*SM*87] "Special issue on causal and diagnostic reasoning." *IEEE Transactions on Systems, Man, and Cybernetics*, **SMC-17** 3 (May/June 1987).

[*SS*77] Stallman, R. and Sussman, G. (1977) "Forward reasoning and dependency-directed bactracking in a system for computer-aided circuit analysis." *Artificial Intelligence*, **9** pp. 135-196.

[*Wo*67] Wos, L., Robinson, G., Carson, D. and Shalla, L. (1967) "The concept of demodulation in theorem proving." *Journal of the Association for Computing Machinery*, **14** pp. 698-709.

[*WO*84] Wos, L., Overbeek, R., Lusk, E. and Boyle, J. (1984) *Automated Reasoning Introduction and Applications*, Prentice-Hall, Englewood Cliffs, NJ.

[*Wo*88] Wos, L. (1988) *Automated Reasoning: 33 Basic Research Problems*, Prentice-Hall, Englewood Cliffs, NJ.

[*WR*69] Wos, L. and Robinson, G. (1969) "Paramodulation and theorem-proving in first order logic with equality." *Machine Intelligence*, Volume 4, American Elsevier, New York, pp. 135-151.

VITA

Barbara Anne Smith was born on December 7, 1956 at Saint Louis, Missouri. She graduated from Rosati-Kain High School in Saint Louis in May, 1973.

She received her undergraduate education at Saint Louis University and in May, 1976 was awarded a B.A. in Mathematical Computer Science, Magna cum Laude. While at Saint Louis University she was elected to membership in Phi Beta Kappa and Pi Mu Epsilon.

From August, 1976 to December, 1977 she was a graduate student in Computer Science at the University of Missouri-Rolla in Rolla, Missouri. During that time she was a teaching assistant for the Mathematics Department at the University of Missouri-Rolla. From January, 1978 to August, 1981 she held positions as a computer programmer at Missouri-Pacific Railroad in Saint Louis and as a systems analyst at Mallinckrodt Institute of Radiology of Washington University Medical School in Saint Louis. She was awarded the Master of Science degree in Computer Science in May, 1980.

From August, 1981 until May, 1985 she held the position of Assistant Professor of Computer Science at Westminster College in Fulton, Missouri.

Since August, 1985 she has been a graduate student at the University of Missouri-Rolla, pursuing the Ph.D. degree in computer science. During that time she has received a Chancellor's Fellowship and has held positions as a teaching assistant and a research assistant.