01 Jan 1989

Fault Diagnosis Using First Order Logic Tools

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

Barbara A. Smith

Recommended Citation

R. W. Wilkerson and B. A. Smith, "Fault Diagnosis Using First Order Logic Tools," *Proceedings of the 32nd Midwest Symposium on Circuits and Systems, 1989*, Institute of Electrical and Electronics Engineers (IEEE), Jan 1989.
The definitive version is available at https://doi.org/10.1109/MWSCAS.1989.101851

Fault Diagnosis using First Order Logic Tools

Barbara A. Smith
Department of Computer Science
University of Dayton
Dayton, OH 45469

Ralph W. Wilkerson
Department of Computer Science
University of Missouri-Rolla
Rolla, MO 65401

Abstract

While numerous diagnostic expert systems have been successfully developed in recent years, they are almost uniformly based on heuristic reasoning techniques (i.e. shallow knowledge) in the form of rules. This paper reports on an automated circuit diagnostic tool implementing Reiter's theory of diagnosis based on deep knowledge (i.e. knowledge based on certain design information) and using first order logic as the representation language. In this approach, the automated diagnostician uses a description of the of the system structure and observations describing its 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. In particular, Reiter's method finds all combinations of components which explain this behavior.

The inference mechanism which is incorporated as part of the diagnostic tool is based on bidirectional constraint propagation. 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. However, 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. A prototype version of the diagnostic program which finds all diagnoses has been developed and successfully demonstated on several small but nontrivial combinational and sequential circuits.

1 Diagnosis from first principles

Raymond Reiter [*Re87*] has developed what he terms a theory of diagnosis. The goal of the theory development is to establish a firm foundation on which to develop automated diagnosticians. His theory is most general and is applicable to many areas of diagnosis. However, the focus of this paper is the diagnosis of circuit faults so 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. De Kleer and Williams [*DW87*] also refer to Reiter's theory as unimplemented. Part of the work of this paper involves an implementation of a diagnostician based on Reiter's theory.

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 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.

In the following section, this terminology is expanded and rephrased within the context of Reiter's general theory. The definitions are taken from [*Re87*]. We begin with the concept of a system which is to be diagnosed. This concept 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.

The name and type of each component is specified in components and gate types. The interconnections of the components are given, as are the observed input and output values. This type of constraint might be absent or quite general, depending on the device to be diagnosed. For example, values might be constrained to be integer or positive in some application or without any constraint in another application. The correct behavior of each component as a function of its input(s) is described by the gate descriptions.

The generality of the approach and representation does not preclude the use of domain specific information concerning faults. 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:

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

Also necessary for diagnosis is one or more sets of observations of the system. An *observation* is simply defined to be a finite set of first order sentences. 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

89CH2785-4/90/0000-0289$01.00 © 1990 IEEE

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

is consistent. A slightly simpler characterization of a diagnosis for (SD, COMPONENTS, OBS) is a minimal set Δ such that $SD \cup OBS \cup \{\neg AB(c) \mid c \in COMPONENTS - \Delta\}$ 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. 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 [dK76]. A *conflict set* for (SD, COMPONENTS, OBS) is a set $\{c_1, \ldots, c_k\}$ 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). Define a minimal hitting set as follows:

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

The following theorem, which Reiter proves, ties together the concepts of minimal hitting sets, conflict sets and diagnoses.

> Theorem: $\Delta \subseteq$ COMPONENTS is a diagnosis for (SD, COMPONENTS, OBS) iff 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 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.

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

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

Reiter states that $H(n)$ for a node n labeled by $\surd$ is a hitting set for F and each minimal hitting set for F is $H(n)$ for some node n for which $\surd$ is the label. Since only minimal hitting sets are desired, a pruned HS-tree will be constructed in such a way that $H(n)$ for any node labeled by $\surd$ is a minimal hitting set.

Some concerns arise in applying the concept of an HS-tree to the process of computing diagnoses. First, the collection of sets F is not explicitly known. In the diagnostic process, F is the set of conflict sets for (SD, COMPONENTS, OBS). Secondly, the determination of a set $f \in F$ is computationally expensive since f is computed by the reasoning component. Thus 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 $S \in F$, 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 F. The label of node n' is underlined to indicate that it is a reused 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 F, 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. A redundant edge, together with the subtree beneath it, may be removed from the HS-tree with preserving the property that the resulting pruned HS-tree will yield all minimal hitting sets for F. A redundant edge in a pruned HS-tree is indicated by cutting it with ")(".

In the context of the diagnostic process, an access to F 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 values. If there exists a conflict set S such that $H(n) \cap S = \{\}$, then the reasoning component must return S otherwise the value $\surd$ is returned. Thus 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. It should be noted again that the underlying reasoning component must be a decision procedure for determining consistency. In general, such decision procedures do not exist. However, in some domains, decision procedures do exist.

Single fault diagnoses are determined by the nodes labeled with $\surd$ at level 1 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.

2 Inference Based on Constaint Propagation

Reiter's theory is independent of the implementation of the underlying inference mechanism. The only requirement is that the inference mechanism be a decision procedure for the domain of the diagnostic problem. In general, a refutational theorem prover is a semi-decision procedure.

The procedure for building the hitting set tree uses all of the pruning techniques suggested by Reiter. These are the reuse of node labels, closing of nodes, and the removal of redundant edges. The last is not necessary if the inference mechanism returns only minimal conflict sets. We have no such guarantee of minimality, however. In addition, we have implemented two heuristics which can further decrease the number of conflict sets which must be computed.

Reusing the label of an existing node in the HS-tree as the label for a new node saves a call to the underlying theorem prover. As the HS-tree grows, there may be several labels which could be reused. The label with the smallest number of elements is chosen. This can be a valuable heuristic since every element of the set labeling a node generates a node at the next level. Further, after a conflict set is returned by the inference mechanism, a check is made to determine whether that conflict set could be used to label an existing node which has not yet been expanded. If the new label would have fewer elements, the node is relabeled.

Our experience has shown that the removal of redundant edges usually does not decrease the number of calls to the theorem prover. Typically, those nodes which are removed by the pruning are closed or can be relabeled and thus do not require that a conflict set be computed. Whether this is an anomaly of the circuit examples we have studied or a feature of the general diagnostic domain is unknown. However, the overhead incurred by the removal of redundant edges is very small compared to the computation cost of even one unnecessary invocation of the theorem prover.

Consequently, our diagnostic program consists of essentially two parts. The first consists of those routines which implement the construction of the pruned HS-tree of Reiter, including the heuristics discussed above. The diagnoses of the given circuit will be read from the pruned HS-tree when it is completely constructed. A diagnosis is the collection of edge labels on the path from the root to a node in the HS-tree which is labeled by $\surd$.

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.

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 $X = SD \cup OBS \cup \{\neg AB(c) \mid c \in C \subseteq COMPONENTS\}$. If the system description is correct, then X is consistent. An inconsistency in X will occur when a behavior predicted by X 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.

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 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 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.

3 Implementation Details

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 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.

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.

REFERENCES

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

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

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

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

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

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