Experience Learning in Model-Based Diagnostic Systems

Yoshiyuki Koseki*
C&C Systems Research Laboratories
NEC Corporation
4-1-1 Miyazaki, Miyamae-ku,
Kawasaki, 213 JAPAN

Abstract

This paper describes a model-based diagnostic system architecture which has test generation capability and which learns from experience. It acquires experiential knowledge from single experiences in the form of compact symptom-failure association rules and component failure records. With this capability, when it has had a similar experience in the past, it can diagnose the failure faster and more efficiently by suggesting better tests. Even if it has not had any similar experience in the past, it can diagnose the failure using its knowledge about the structure and behavior of the target system. Therefore, this architecture gives a solution to both the knowledge acquisition bottleneck problem of heuristic rule based systems and the efficiency problem of model-based systems. Experimental results show this technique to be practical and promising. It has been implemented in PROLOG in a concise form.

1 Introduction

Advances in modern design and manufacturing technology have enabled us to build devices of high complexity. When those devices fail to function correctly, they need to be diagnosed for faulty components. Because the complexity of diagnosis increases with increasing design complexity, the efficient automation of this task is essential.

There have been two distinct approaches to automate the diagnostic task. One is based on *heuristic-based* expert system methods, such as the one in the MYCIN system [Shortliffe, 1976], The other is based on *model-based* methods, i.e. *first-principle* methods. Heuristic-based expert system methods use symptom-failure association rules gathered by interview from experts experienced in a domain. Model-based methods use design descriptions, such as the structure and behavior descriptions.

Heuristic-based expert systems can guide efficient diagnosis for known cases. However, they lack robustness because they cannot deal with unexpected cases not covered by heuristic rules. Their knowledge bases are expensive to create, since interviewing requires substantial effort, as does coding and debugging of the knowledge base. Maintenance is also difficult, and even a small design change in the target device may require thorough review of the knowledge base.

Model-based methods include conventional Boolean-Logic-level diagnostic methods, such as D-algorithrn [Roth *et al*., 1967]. Since they are specialized in Boolean logic-level, they suffer from a scaling problem in dealing with modern complex devices which have millions of logic-level subcomponents. Recently introduced model-based methods, such as [Genesereth, 1984], [Davis, 1984], [deKleer and Williams, 1987], [Reiter, 1987], and [Poole, 1986], provide declarative device-independent design representation languages and device-independent diagnostic procedures. As a consequence of this generality, they are capable, utilizing hierarchical designs, of diagnosing complex devices. Moreover, they can be used for wider classes of devices, including non-digital and non-electronic devices. They are more robust than heuristic-based systems, because they can deal with unexpected cases not covered by heuristic rules. Their knowledge bases are less expensive to create and flexible in regard to design changes since they are a straightforward representation of designs which are likely to be found in modern CAD environments. They do not require rule verification, which can be a serious problem in writing heuristic rules. They can provide a working diagnostic system when the target device is ready for delivery because they do not require any field experience for building the knowledge base.

However, model-based diagnostic systems are generaly not as efficient as heuristic-based ones since they require more complex computation. Furthermore, they are not always able to pinpoint a failing component from the available symptom information and sometimes require many tests to reach a conclusive decision. This is because they lack heuristic knowledge.

This paper introduces a model-based diagnostic system architecture with test generation capability, one which is able to learn from its experience to improve its performance incrementally. It is basically model-based, and it learns heuristic knowledge from experience in terms of *symptom-failure association rules* and *com-*

*This work was mainly done while the author was at the Computer Science Department of Stanford University as a visiting scholar for the 1987-1988 academic year.

ponent failure models. It also caches general rules for model-based diagnosis and test pattern generation for efficient computations.

In this paper, the author is making a *single fault assumption.* That is, it is assumed that there is only one malfunctioning component in a failing device. This assumption makes it easy to rule out certain failure hypotheses using test results. The author also makes a *non-intermittency fault assumption.* That is, it is assumed that the behavior of a failing device does not change during the diagnosis process. This enables localizing a fault by testing the device after the symptom appears. In addition, the method is based on the nature of the *fault locality.* It is usually true that most of the faults which occur in a device are local to some particular subcomponents. That is to say, a failed subcomponent tends to fail again in a similar manner in the future. This supports the use of fault models and symptom-failure association rules which are learned by analyzing past experience.

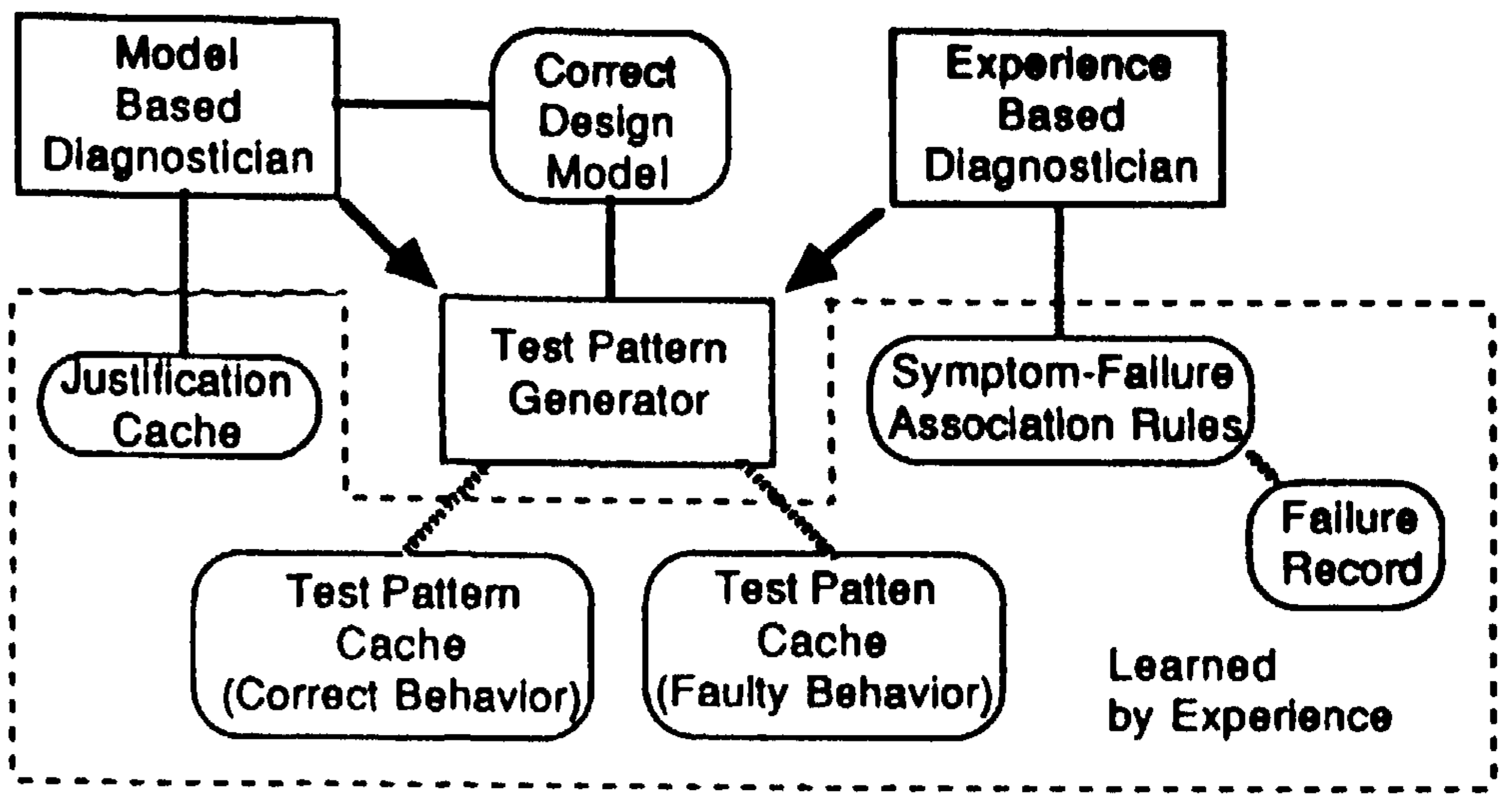
This paper is organized as follows. The first section to follow uses a simple example to present the general structure and general flow of the proposed diagnostic system. The second section explains inference procedures. Experimental results, indicating promising data, are shown in the third section. The last section evaluates and discusses the utility of the approach.

2 Architecture

2.1 Structure

A generic diagnostic system architecture described here is based on a previously reported study of the knowledge acquisition task of an expert system for telephone switching system diagnosis [Koseki *et al.*, 1987]. Maintenance experts can quickly identify a faulty component if they have solved a similar problem in the past. If a novel symptom arises, the expert consults with the design description manuals to find out how the system is supposed to work, and what components might have gone wrong which would have caused this failure to occur. Then he performs several tests to localize the fault and repairs the failing component. Interestingly, when he again confronts a similar case in the future, he can diagnose the failure using fewer tests without consulting the design description.

The proposed architecture structure is shown in Figure 1. The *model-based diagnostician* diagnoses the failure with the *correct design model,* which includes description of the structure and behavior of the target device. The *experience-based diagnostician* uses experiential knowledge represented in the form of heuristic *symptom-failure association rules.* If some faulty subcomponent behavior is known to be probable, symptom-failure association rules for such behaviors may be provided in advance by using the *test pattern generator.* The *test pattern generator* generates tests to discriminate failing components from working components. The model-based diagnostician caches its inference reasoning result in the *justification cache,* which is a set of generalized compact rules based on previously experienced cases. The test pattern generator also caches generated test patterns in a *test pattern cache.* This architecture is novel in the sense that it utilizes both model-based and experience-based diagnosticians, and it acquires knowledge incrementally from past experience.



Model Based Diagnostician
Correct Design Model
Experience Based Diagnostician
Justification Cache
Test Pattern Generator
Symptom-Failure Association Rules
Failure Record
Test Pattern Cache (Correct Behavior)
Test Patten Cache (Faulty Behavior)
Learned by Experience

Figure 1: General Structure

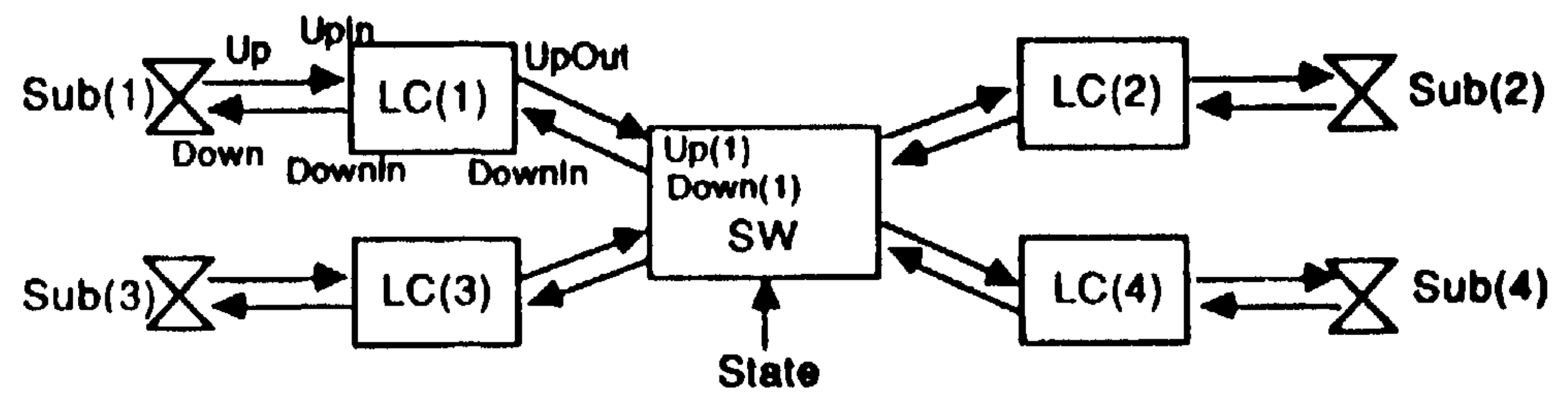


Figure 2: Example Device

2.2 Design Representation

This section summarizes the design model representation. A simplified telephone switching system, shown in Figure 2, is used as an example. All descriptions are expressed in the form of first order predicate calculus, with the conventions used in [Genesereth and Nilsson, 1987].[1]

The design description for the system consists of descriptions of its structure and its behavior. The structure description consists of nine subcomponents and interconnections between them. The following facts define the connections. The dash character - is used as an infix operator to represent a port with a component and a subport name. The first line states that output-port Up of subscriber Sub(n) is connected to input-port UpIn of Line Circuit LC(n). The remaining lines describe the other connections in a similar way.

Conn(Sub(n)-Up, LC(n)-UpIn)
Conn(LC(n)-UpOut, SW-Up(n))
Conn(SW-Down(n), LC(n)-DownIn)
Conn(LC(n)-DownOut, Sub(n)-Down)

The behavior descriptions are specified by a set of rules. The connection behavior, with zero time delay, is given

^he lower case letters stand for universally quantified variables, the upper case letters for constants, A for logical conjunction, V for logical disjunction, -* for negation, and <= and =* for logical implication.

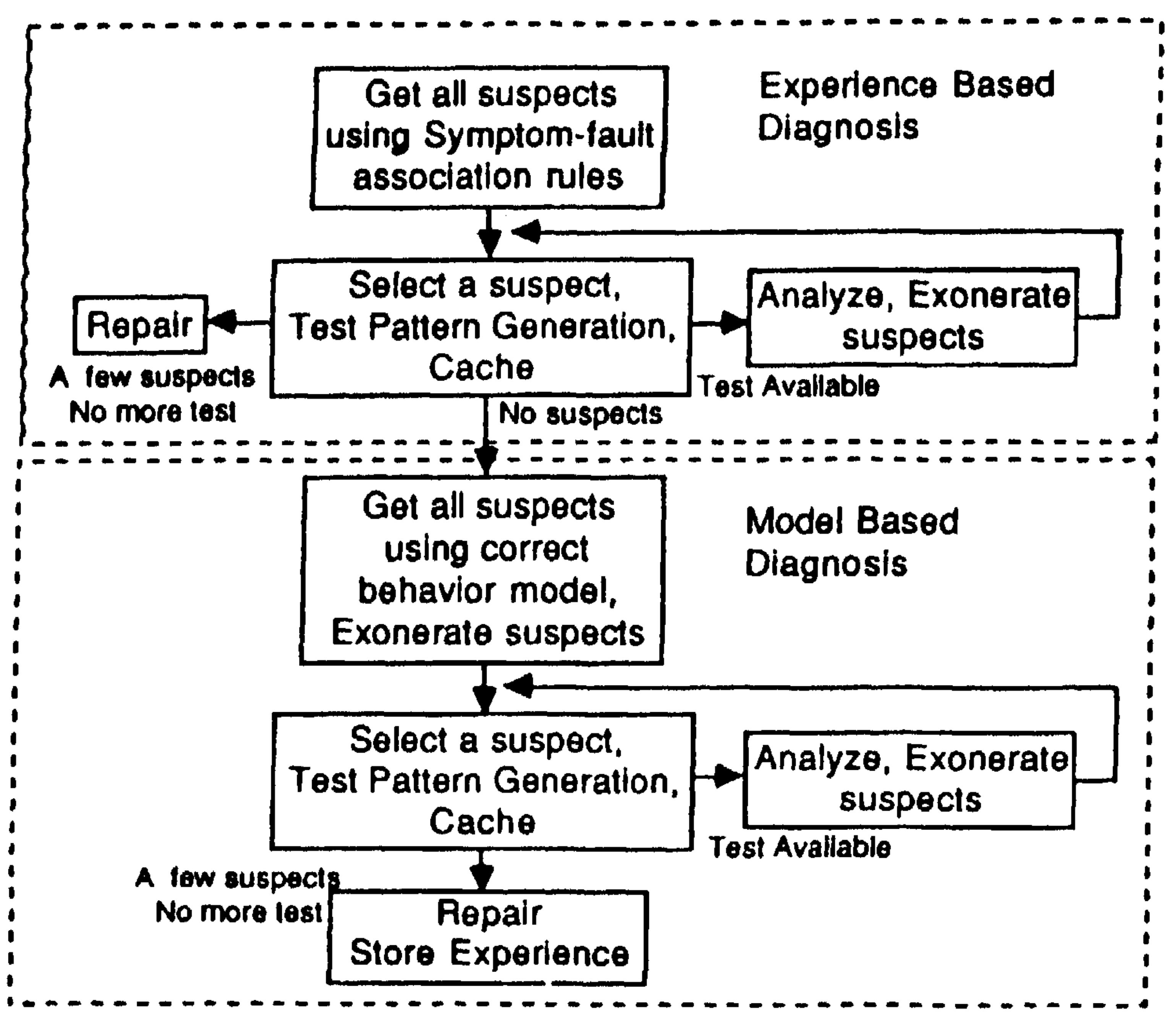

Get all suspects using Symptom-fault association rules
Experience Based Diagnosis
Select a suspect, Test Pattern Generation, Cache
Analyze, Exonerate suspects
Repair
A few suspects No more test
No suspects
Test Available
Get all suspects using correct behavior model, Exonerate suspects
Model Based Diagnosis
Select a suspect, Test Pattern Generation, Cache
Analyze, Exonerate suspects
Test Available
A few suspects No more test
Repair Store Experience

Figure 3: General Flow

by the rule given below. This rule states that port q has value v at time t, if port p is connected to port q, and port p has value v at time t.

$$\mathrm{Val}(q,v,t) \Leftarrow \mathrm{Conn}(p,q) \wedge \mathrm{Val}(p,v,t)$$

The behavior descriptions for the subcomponents are given by the rules listed below. For simplicity, it is assumed that all of the subcomponents have zero time delay. The first rule states that output Down of switch SW is x at time t, if input State of SW is nl-n2, input Up of SW is x, and SW is working correctly for inputs State and Up and output Down. The remaining rules state the behaviors for Line Circuit LC.

$$\begin{aligned}&\mathrm{Val}(\mathrm{SW\text{-}Down}(n2),x,t) \Leftarrow\\&\quad \mathrm{Val}(\mathrm{SW\text{-}State},n1\text{-}n2,t) \wedge \mathrm{Val}(\mathrm{SW\text{-}Up}(n1),x,t)\\&\quad \wedge \mathrm{Working}(\mathrm{SW},[\mathrm{State}(n1\text{-}n2), \mathrm{Up}(n1,x)], \mathrm{Down}(n2,x))\\&\mathrm{Val}(\mathrm{LC}(n)\text{-}\mathrm{UpOut}, x, t) \Leftarrow \mathrm{Val}(\mathrm{LC}(n)\text{-}\mathrm{UpIn}, x, t)\\&\quad \wedge \mathrm{Working}(\mathrm{LC}(n), [\mathrm{UpIn}(x)], \mathrm{UpOut}(x))\\&\mathrm{Val}(\mathrm{LC}(n)\text{-}\mathrm{DownOut}, x, t) \Leftarrow \mathrm{Val}(\mathrm{LC}(n)\text{-}\mathrm{DownIn}, x, t)\\&\quad \wedge \mathrm{Working}(\mathrm{LC}(n), [\mathrm{DownIn}(x)], \mathrm{DownOut}(x))\end{aligned}$$

2.3 General Flow

The general flow of the diagnostic system is shown in Figure 3. The system first tries experience-based diagnosis, and if it fails to diagnose the given symptom, it performs model-based diagnosis. The experience-based diagnostician first obtains a list of suspects, using the previously learned symptom-failure association rules. The obtained suspect list is sorted in order of failure probability, according to the failure records. This ordering gives higher priority in testing to the more probable failures than for the less probable failures. When the number of suspects is greater than one, a test is generated by the test pattern generator for the failure remaining at the top of the suspect list. The test result is then analyzed to exonerate the non-failure components. This test-and-analyze cycle is repeated until the number of suspects is significantly reduced or until there is no test available. The system then suggests a repair of the remaining suspects. If this experience-based diagnosis phase fails to identify the faulty components, i.e. the suspect list becomes empty or the problem is not fixed by the suggested repair, a model-based diagnosis is conducted.

The model-based diagnostician works similarly, except that it does not use any failure models. When it gets a suspect list for a given symptom, it caches a compact generalized rule in the *justification cache.* It also iterates a test-and-analyze cycle until it gets small number of suspects or runs out of available tests. Every time it generates a test, it caches the test pattern for future use. After successfully repairing the faulty component, the system learns from the case. It analyzes the symptom and identifies the faulty behavior of the failing component. It then stores a generalized symptom-failure association rule based on that analysis.

In the example switching system, suppose that when Sub(l) made a call to Sub(2), noise was heard in addition to the voice at Sub(2) at time 100. The symptom may be described as:

$$\mathrm{Val}(\mathrm{SW\text{-}State}, [1\text{-}2], 100) \wedge \mathrm{Val}(\mathrm{Sub}(1)\text{-}\mathrm{Up}, \mathrm{Voice}, 100) \wedge \mathrm{Val}(\mathrm{Sub}(2)\text{-}\mathrm{Down}, \mathrm{Noise}, 100)$$

Suppose that the system has no initial experiential knowledge. The experience-based diagnostician would then fail to identify the failure, and the system would perform a model-based diagnosis. The model-based diagnostician gives a list of suspects in terms of the subcomponent working-conditions which should have worked correctly. In our example here, it gives a list in a disjunction of negated subcomponent working conditions:

$$\begin{aligned}&\neg\mathrm{Working}(\mathrm{LC}(1), [\mathrm{UpIn}(\mathrm{Voice})], \mathrm{UpOut}(\mathrm{Voice})) \vee\\&\neg\mathrm{Working}(\mathrm{SW},[\mathrm{State}(1\text{-}2),\mathrm{Up}(1,\mathrm{Voice})],\mathrm{Down}(2,\mathrm{Voice}))\vee\\&\neg\mathrm{Working}(\mathrm{LC}(2), [\mathrm{DownIn}(\mathrm{Voice})], \mathrm{DownOut}(\mathrm{Voice})).\end{aligned}$$

Note that the order of components in the list is arbitrary. A test pattern then is generated for the component behavior on the top of the list. It is assumed here that all the observable points and the controllable points are subscriber ports. In this case, the test pattern generator gives a test pattern to test LC(1) behavior, which inputs Voice to Sub(l)-Up and tries to observe Voice at Sub(4)-Down. Suppose that noise was observed again. This gives another set of suspects, that is,

$$\begin{aligned}&\neg\mathrm{Working}(\mathrm{LC}(1), [\mathrm{UpIn}(\mathrm{Voice})], \mathrm{UpOut}(\mathrm{Voice})) \vee\\&\neg\mathrm{Working}(\mathrm{SW},[\mathrm{State}(1\text{-}4),\mathrm{Up}(1,\mathrm{Voice})],\mathrm{Down}(4,\mathrm{Voice}))\vee\\&\neg\mathrm{Working}(\mathrm{LC}(4), [\mathrm{DownIn}(\mathrm{Voice})], \mathrm{DownOut}(\mathrm{Voice})).\end{aligned}$$

By the single fault assumption and the nonmtermittency assumption, those two suspect sets are merged to one suspect list [Singh, 1988]:

$\neg$Working(LC(1), [UpIn(Voice)], UpOut(Voice)) $\vee$
($\neg$Working(SW,[State(1-2),Up(1,Voice)],Down(2,Voice))
$\wedge$
$\neg$Working(SW,[State(1-4),Up(1,Voice)],Down(4,Voice))).

Next, a voice is input to Sub(3)-Up and a check is made for a voice at Sub(4)-Down. Suppose that a normal output voice is observed. This exonerates the second disjunction for SW in the suspect set.[2] After verifying that the diagnosis was correct by repairing or replacing the LC(1), the system learns a generalized, compact symptom-failure association rule for the initial symptom, in the following form:

if Val(Sub(n2)-Down, Noise, t) $\wedge$
Val(SW-State,[n1-n2],t) $\wedge$ Val(Sub(n1)-Up,Voice,t)
then Failed(LC(n1), [UpIn(Voice)], UpOut(Noise)).

The system also records the failure information in the form of Failed(LC(n1), [UpIn(Voice)], UpOut(Noise), 1). The last argument is the failure number counter and it holds the number of the times the system has experienced this fault in the past. This information represents the failure probability and will be used by the experience-based diagnostician for sorting the suspect list.

After the system has learned from a first experience, it is capable of dealing with similar symptoms more efficiently. Suppose that there is another similar but different symptom, such as when Sub(3) made a call to Sub(4), noise was heard at Sub(4) at another time. This symptom matches the previously learned association rule, and produces a suspect list with only one suspect, which is Failed(LC(3), [UpIn(Voice)], UpOut(Noise)). The test pattern generator generates a test to check this faulty behavior. It tries to observe Noise at Sub(2). If this faulty behavior is confirmed, it is concluded that LC(3) is faulty and producing a noisy output, and the system suggests a repair to fix it.

3 Inference Procedures

The diagnosis inference and the test pattern generation is based on an abduction inference method called *resolution residue* [Finger and Genesereth, 1985] which is used in DART [Genesereth, 1984] and SATURN [Singh, 1987]. This method is functionally equivalent to the *constraint suspension* technique of [Davis, 1984] and the Theorist framework of [Poole, 1986]. This inference procedure uses a domain theory, a goal, and a specification of facts called *assumablcs,* which can be assumed to prove the goal. It deduces a set of assumable facts which, together with the domain theory, entails the goal. To learn symptom-failure association rules and to generate generalized rules for caching, the residue inference method is extended using *Explanation Based Generalization* (EBG) techniques [Mitchell *et al*., 1986, Dejong and Mooney, 1986]. The extended inference procedure is called REBG.

EBG method uses a *goal concept* and a *training example* with the *domain theory* to compose an explanation tree, which explains why the training example is an instance of the goal concept, given the domain theory. The explanation tree is then generalized to obtain a general concept for the goal in terms of *operational criteria.* In other words, this method performs two functions. First, it examines the given inference explanation and re-expresses it in a compact rule form in terms of operational criteria. Performance gain here is mainly attributable to this function. The EBG method then generalizes the obtained rule by replacing instantiated variables by as many variables as possible. This function makes the learned rule applicable to other similar cases. REBG performs those functions in residue computation.

3.1 Model-Based Diagnosis

The input to the model-based diagnosis procedure is a design description *D* and a symptom description < /,->0 >, where / is a conjunction of input (controllable) port values and *O* is an expected output (observable) port value. The model-based diagnostician produces a set of possible failures (suspects) in terms of working-conditions *W* for the components. In applying the residue procedure to a model-based diagnosis problem, the domain theory corresponds to design description *D* and input port values /. Assumable facts specify working conditions *W* for the components, and the goal is output port value *O.* A set of assumable working conditions *W* is then computed so that $I \wedge D \wedge W \Rightarrow O$ is true. This is equivalent to $I \wedge \neg O \wedge D \Rightarrow \neg W$. Since design *D* is assumed to be correct, this formula corresponds to the diagnosis rule $I \wedge \neg O \Rightarrow \neg W$. Note that this procedure does not require any subcomponent failure models. This design was considered important because of the extreme difficulty of assuming failure models, such as stuck-at-faults, in general devices.

In the previous example, *I* and *O* are as follows:

I: Val(SW-State, [1-2], 100) $\wedge$ Val(Sub(1)-Up, Voice, 100)
O: Val(Sub(2)-Down, Voice, 100)

Note that *O* is the expected correct value instead of the actual failing value. The assumable facts specify the working conditions for components and have the following form:

Working(<component>, [<input-value>, ...], <output-value>).

With this procedure, working conditions *W* are computed for this form, which, together with design *D* and input /, entail output *O.* The procedure employs the depth-first directed resolution strategy of [Genesereth and Nilsson, 1987]. It works backward to prove the goal by assuming some of the assumable facts. It concludes the working conditions *W* for the output *O* to be correct:

[2]Under an assumption that if an instantiated behavior of a component is correct, the behavior rule in general is correct. Although this assumption is not always true, it is not a relevant problem to the discussion here. A more formal treatment of this situation is given in [Singh, 1988].

Working(LC(1), [UpIn(Voice)], UpOut(Voice)) ∧
Working(SW, [State(1-2), Up(1,Voice)], Down(2,Voice)) ∧
Working(LC(2), [DownIn(Voice)], DownOut(Voice)).

Negation of this proposition produces the desired suspect list.

3.2 Knowledge Caching by REBG

Model-based reasoning is expensive because it involves finding a proof for the correct output value by examining the structure and behavior of the device. One way to avoid this computation is to compile all of the cases into symptom-failure association rules in advance. This method may be feasible if probable subcomponent misbehaviors are known in advance.

As an alternative to compiling all the cases, this system partially compiles and caches its model-based knowledge into compact generalized rules for the cases it has experienced. The REBG procedure generates such rules while it computes residue. It is essential to store *generalized* rules rather than *instance* rules in order to cover the entire class of symptoms related to cases experienced.

3.3 Association-Rule Generation by REBG

After a successful model-based diagnosis, REBG also generates symptom-failure association rules to be used by the experience-based diagnostician. REBG computes a failing component behavior, which can be assumed to prove the faulty output. This procedure generates a compact generalized rule at the same time to cover the entire class of related cases. In this case, the assumable facts are the behavior of the failing component.

3.4 Test-Pattern Generation

The test-pattern generation algorithm is based on the work of SATURN system [Singh, 1987]. It is an algorithmic test generation system, similar to the D-algorithm. However, it works on designs specified at arbitrary abstraction levels rather than at the Boolean logic level. The procedure takes as inputs the design description *D* and the subcomponent input-output behavior to be tested < *Is,Os* >. It propagates *Is* to a set of controllable input values *I*, and *Os* to a set of observable output values O, by using the residue method. To increase efficiency, it utilizes the control strategies of *conditional values* and *consistency checking.* Conditional values are used to propagate only useful values to the observable output ports. Consistency checking is used to eliminate inconsistent alternatives in earlier stages of the search process.

The experience-based diagnostician uses this algorithm to propagate a *faulty* behavior of the suspected subcomponent, while the model-based diagnostician propagates a *correct* behavior. Since this computation is also expensive, after it generates a test pattern for a suspect, the system caches the result for future use.

4 Experimental Results

An experimental system was implemented as a PROLOG meta-interpreter on a DEC-20 computer, using the EBG implementation technique described in [Kedar-Cabelli and McCarty, 1987, Hirsh, 1988]. With general resolution residue procedures, assumed facts must be checked against the theory for consistency, a task which is not ordinarily semi-decidable, but fortunately, this task is not necessary here for caching and generating association rules. This is because it is obvious that assuming the correct behaviors of components does not cause any inconsistency with other rules in the theory.

The system was tested on several designs including the example switching system. In the example mentioned in the previous section, the number of tests required to identify a fault by the model-based diagnostician was three, while the experience-based diagnostician required only one test.

The performance improvement achieved with knowledge caching for a model-based diagnosis is shown in Table 1. It shows CPU times for a full-adder circuit, a circuit called D74, which computes a sum of products[3], the example telephone switching system SW, and a 7-segment decoder circuit 7SEG. The first and second rows, labeled SUM and COUT, are for the two outputs from full-adder circuit. The CPU times are measured in *msec* for the compiled meta-interpreter. The first column shows the time used in the normal model-based diagnosis. The second column is for the enhanced procedure REBG, which produces generalized rules at the same time. The third column shows the actual performance improvement achieved with cached rules. These examples show that the REBG function is approximately five times slower than the regular residue procedure. Improvement with the cached rules is by a factor of ten.

In the circuits with some reconvergent fanout points, simple backward inference may include redundant computation of the same values. This redundancy can be checked and eliminated in the generalization process. When the system assumes an assumable fact, it checks its uniqueness against previously assumed facts. Similarly, when the system composes a rule with operational facts, it checks its uniqueness against previously used operational facts. Redundant facts are eliminated and not included in the learned rules. A 7-segment decoder contains many fanout reconvergent points. Redundancy checking was effective in eliminating redundancy in generated rules for this circuit. Using the rules generated without redundancy checking, it took 24 ms to compute suspects, while it took 10 ms when using rules with redundancy checking.

5 Discussion

When the experience-based diagnostician has had a similar experience in the past, it can diagnose the failure faster and more efficiently by suggesting better tests. The number of tests to discriminate one suspect out of *n* suspects is $\log_2 n$, when every test is chosen to reduce the number of suspects by half. It is *n* in the worst case, and 1 in the best case. For target devices which have a fault locality, the number of suspects enumerated by the

[3]The circuit is also used in [Gcnesereth, 1984] and [Davis, 1984] to describe their systems.

	Model-based Diagnostician (msec)		Cached-Rules (msec)
	Residue	REBG	
SUM	27	141	3
COUT	49	243	5
D74	34	137	4
SW	26	271	5
7SEG	500	1990	10

Table 1: Performance Improvement by Cached Rules

experience-based diagnostician is kept small. Since the number of tests to localize a fault depends on the number of suspects, the experience-based diagnostician may find faults with fewer tests than will the model based diagnostician.

The experimental results show that caching improves efficiency significantly. It has been pointed out that when too many rules are learned, the caching technique does not always improve the efficiency. It is true that the larger the cache becomes, the longer it takes to find an appropriate rule. However, even in the worst case, where all of the possible combinations of the symptoms are cached, the complexity of computing suspects by learned rules never exceeds the complexity involved when using the model-based diagnostician. This is because the learned rules are simply compact forms of an inference chain, obtained in the model-based inference. There is not much indexing overhead in looking up the learned rules because the rules are indexed by their first arguments which hold output port names.

To limit the performance degradation which will be caused by an increase in the number of learned rules, only frequently used rules should be retained. Those which have not been used for a long time may be forgotten. This corresponds to the strategy described in Minton, 1988] for using a utility function. This treatment is considered to be practical in the diagnosis domain because of its tendency to have fault locality; that is, components which have once failed in the past tend to fail again and similar symptoms are likely to reappear.

This proposed method has a notable advantage in correctness. While it has a comparable efficiency to conventional rule-based expert system approaches, it provides a sound and complete diagnosis. Therefore, it solves the serious rule verification problems of conventional approaches. Since learned rules are generated from the behavior and structure model of the target equipment, they are guaranteed to be sound. A set of learned rules in the experiential knowledge base may be incomplete, but the entire diagnosis system gives a complete diagnosis because the symptoms not covered by the experience-based diagnostician are diagnosed by the model-based one.

6 Conclusion

This paper has described a model-based diagnostic system architecture with test generation capability, one which learns from experience. Its architecture offers a solution to the two main problems encountered in heuristic-rule based systems, brittleness and knowledge acquisition bottlenecks. It also solves the efficiency problem in model-based systems. Future testing of the system on an actual communication system is in the planning stages. The general residue inference procedure and the learning procedure used in the system are currently incorporated into a PROLOG based knowledge programming framework PEACE [Koseki, 1987]. At present, the system works on one level of abstraction. Future work is to include the use of hierarchical design models for diagnosis and for test pattern generation.

Acknowledgments

The author would like to thank Prof. Michael Genesereth for giving him the opportunity to work at Stanford, and for his technical suggestions throughout the work. He thanks Narinder Singh for his technical assistance in depth and his thorough review of an early draft of this paper. This work also has benefited greatly from Haym Hirsh's introducing to us the Prolog implementation of EBG, and valuable comments on a draft of this paper from Devika Subramanian, Stuart Hamilton, and Mark Petersen. Finally, the author wishes to express his appreciation to Yasuo Kato, Tatsuo Ishiguro, Satoshi Goto, and Hajimu Mori of NEC Corporation for giving him the opportunity to pursue this research at Stanford, and to Shin-ichi Wada for his support.

References

[Davis, 1984] R. Davis. Diagnostic Reasoning Based on Structure and Behavior. *Artificial Intelligence,* 24:347 410, 1984.

[Dejong and Mooney, 1986] G. Dejong and R. Mooney. Explanation-Based Learning: An Alternative View. *Machine Learning* 1:145-176, 1986.

[deKleer and Williams, 1987] J. de Kleer and B. C. Williams. Diagnosing Multiple faults. *Artificial Intelligence,* 32:97-130, 1987.

[Finger and Genesereth, 1985] J. J. Finger and M. R. Genesereth. A Deductive Approach to Design Synthesis HPP-85-1, Stanford University Heuristic Programming Project, January, 1985.

[Genesereth, 1984] M. R. Genesereth. The Use of Design Descriptions in Automated Diagnosis. *Artificial Intelligence,* 24:411-436, 1984.

[Genesereth and Nilsson, 1987] M. R. Genesereth and N. J. Nilsson. *Logical Foundation of Artificial Intelligence.* Los Altos, CA: Morgan Kaufmann, 1987.

[Hirsh, 1988] H. Hirsh. Reasoning about Operationally for Explanation-Based Generalization. In *Proceedings of the Fifth International Machine Learning Conference,* 1988.

[Kedar-Cabelli and McCarty, 1987] S. T. Kedar-Cabelli and L. T. McCarty. Explanation-Based Generalization as Resolution Theorem Proving. In *Proceedings* *of the Fourth International Workshop on Machine Learning,* pages 383-389, 1987.

[Koseki *et al,* 1987] Y. Koseki, S. Wada, T. Nishida, and H. Mori. SHOOTX: A Multiple Knowledge Based Diagnosis Expert System for NEAX61 ESS. In *Proceedings of the International Switching Symposium 87,* pages 78-82, Phoenix, 1987.

[Koseki, 1987] Y. Koseki. Amalgamating Multiple Programming Paradigms in Prolog. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence,* pages 76-82, Milan, August, 1987.

[Minton, 1988] S. Minton. Quantitative Results Concerning the Utility of Explanation-Based Learning. In *Proceedings of the Seventh National Conference on Artificial Intelligence,* pages 564-569, Saint Paul, August, 1988.

[Mitchell *et al,* 1986] T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli. Explanation-Based Generalization: A Unifying View. *Machine Learning* 1:47-80, 1986.

[Poole, 1986] D. L. Poole. Default Reason ing and Diagnosis as Theory Formation. Technical Report CS-86-08, Department of Computer Science, University of Waterloo, March, 1986.

[Reiter, 1987] R. R eiter. A Theory of diagnosis from first principles. *Artificial Intelligence* 32:57-95, 1987.

[Roth *et al,* 1967] J. P. Roth, W. G. Buricius, and P. R. Schneider. Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits. *IEEE Trans. Electron. Comput.,* vol. EC-16, no. 10:567-580, 1967.

[Shortliffe, 1976] E. J. Shortliffe. *Computer Based Medical Consultations: MYCIN,* Elsevier, New York, 1976.

[Singh, 1987] N. P. Singh. *An Artificial Intelligence Approach to Test Generation.* Norwell, MA: Kluwer Academic, 1987.

[Singh, 1988] N. P. Singh. Generating Diagnostic Procedures for Discrete Devices. Logic-88-7, Logic Group Report, Computer Science Department, Stanford University, August, 1988.