A Design Based Approach to Constructing Computational Solutions to Diagnostic Problems

D. Volovik & I. A. Zualkernan
Department of Computer Science
University of Minnesota, Minneapolis, Minnesota USA

P. E. Johnson
Department of Information & Decision Sciences
University of Minnesota, Minneapolis, Minnesota USA

C. E. Matthews
IBM Application Business Systems, Rochester, Minnesota USA

Abstract

Troubleshooting problems in real manufacturing environments impose constraints on admissible solutions that make the computational solutions offered by "troubleshooting from first principles" and the conventional experience based expert systems approaches infeasible. In this paper we present a computational theory for a solution to these problems that is based on the Principle of Locality and exploits the domain specific weak methods of troubleshooters and debugging knowledge of the designers. The computational theory is evaluated by generating focus of attention heuristics for a moderately complex digital device.

1. Computational Problem

We are interested in finding computational solutions to diagnostic problems as they occur in real manufacturing environments [Johnson 89]. This class of problems introduces two constraints on any computational solution: 1) the devices are large and complex, which precludes any solution that relies on a complete simulation or on enumeration of the various fault propagation paths through one or more abstractions of the device (e.g., [Genesereth 84],[de Kleer 87], [Reiter 87]) and 2) the life-cycle of these devices is short, which makes any solution (e.g., [Freiling 85]) that relies on trouble-shooting knowledge specific to a particular device unfeasible.

The first constraint suggests that any computational solution to this class of problems should have a uniform mechanism for handling the order of complexity of these devices. The second constraint precludes the use of any troubleshooting knowledge that is compiled for a specific device as an adaptation to the task of troubleshooting that device.

2. Computational Theory

The task of troubleshooting complex large-scale devices can be viewed as a two step process: 1) Determine an appropriate search space in which the fault is "local" [Davis 84] and 2) Apply weak-troubleshooting methods on this space to locate the fault. A search space is defined by a specification of appropriate 'pathways of interactions" that are exploited by a weak troubleshooting method.

Although the definition of our computational problem precludes the use of device-specific troubleshooting knowledge, device-independent but *domain-specific* troubleshooting methods [Reed 88] developed by troubleshooters can be a valid component of a computational solution. However, a solution that relies solely on these methods, given the constraints of the computational problem is not adequate. There are two difficulties with such a solution:

1) Domain specific weak methods are used by troubleshooters only when the device specific heuristics fail; this means that the best performance one can expect from a solution that relies solely on these methods would be that of a troubleshooter's performance on a new device.

2) The locus of search spaces that can be used by troubleshooters is restricted by device representations available in their task environment. The representations of a device available to troubleshooters in real manufacturing environments, however, often consist of just the physical device itself (which leads to the used of a search space based on physical pathways of interaction) and at most one other level of abstraction (such as schematic for digital designs). This means that any solution that relies solely on weak-methods of troubleshooting will have the restriction that it can only work for faults that are 'local' in search spaces that can be derived from these representations. A large class of interesting faults are not 'local' in a search space derived from the physical representation of a device [Davis 84].

Another source of knowledge that is admitted by the definition of the computational problem is that of designers conducting the task of debugging, which though different from troubleshooting, has components that are similar to the troubleshooting task. Designer's task environment has a rich variety of representations of the device. In any design process, these consist of various by-products of the design. For example, in the case of digital hardware

design, the designers may use requirements, functional specifications and block level descriptions in addition to the schematics and the physical representation of a device.

Our solution to the computational problem described above is to exploit the knowledge of how designers debug designs in addition to the device-independent methods and search spaces of the troubleshooters. This solution has the advantage that it can detect faults that are not only 'local' in the search spaces of troubleshooters, but ones that are 'local' in the various spaces that result from the combination of the search spaces used by designers for debugging a class of devices.

This solution is based on a computational theory that can be stated as follows:

1) The task of troubleshooting is to determine the appropriate search space in which a fault is local and then apply *domain dependent* weak troubleshooting method to the search space until the fault is found.

2) The source for domain dependent weak troubleshooting methods is the task of troubleshooting in the domain.

3) The source for appropriate search spaces is task of debugging by designers in the domain during the design phase.

The construction of a computational solution based on the computational theory consist of the following steps:

1) Determine weak trouble-shooting methods used by troubleshooters in a domain.

2) Determine search spaces used by designers in debugging designs in the domain

3) Use the "principle of locality" as an organizational principle to construct a solution that applies weak troubleshooting methods of troubleshooters to search space used by the designers.

3. Experimental Validation of the Computational Theory

To asses the validity of the computational theory described above, we chose the domain of small card digital design. This is an appropriate domain under the definition of the computational problem as the small card designs are fairly complex devices (~ 10^2 - 10^3 components at the schematic level) and the life-cycles of these cards are short; typical life is ~ 2 years.

3.1 Constructing a Computational Solution for Digital Devices

Previous work done in the domain of troubleshooting digital devices indicates that troubleshooters use a variety of domain specific weak troubleshooting methods [Reed 89]. As a first test of the computational theory, we concentrate on a method called *initial focus of attention* method. This is a method that troubleshooters use to initially focus on a sub-set of the device, given certain symptoms. The symptoms in the case of digital design typically consist of test failures on a specific device.

As a first step in the validation process, a designer (with 10 years of experience with constructing small card digital designs) was asked to construct a design for an *Interrupt Control Coprocessor* with the following features:

- One level prioritized Interrupts
- Subroutine handling capabilities
- LIFO stack for return address storage
- Interrupt masking by software and hardware
- Stack full flag

The resulting device consisted of 254 components (LSI's and gates) with approximately 10^4 signals. In addition to the designing the device, the designer was also asked to write tests (see endnote #1) for the device (a normal part of the design process). The designer was asked to think aloud as he conducted the design and these comments (verbal protocols) were tape-recorded along with the various by-products generated by the designer as he constructed the design.

An analysis of the verbal protocols during the process of debugging design indicated that the designer used the search space schema for initial focus of attention shown in Figure 1.

This space is based on pathways of interaction that go across the various representations of design available to the designers (e.g., functional specifications, workbook level design and tests).

Given the nature of the search space used by the designers and the domain specific weak troubleshooting methods of troubleshooters, the principle of locality was used to construct a computational model that used "path following" weak method on the search space.

To apply the principle of locality we identify the types of faults that are local in component spaces of the search space. For example, in the space defined by test specifications, immediately obvious faults are patterns of test failures. These failures are detected in the space through observations of test outcomes generated by the testing equipment. Interactions of tests and specifications establish pathways of interaction between these two spaces. If pathways of interaction map local faults in one space (e.g. specification faults) into local faults in another space (e.g. test failures), propagating the faults from one space into another along the pathway is desirable. This allows detection of specification failures in a specification space by observing test failures in the test space.

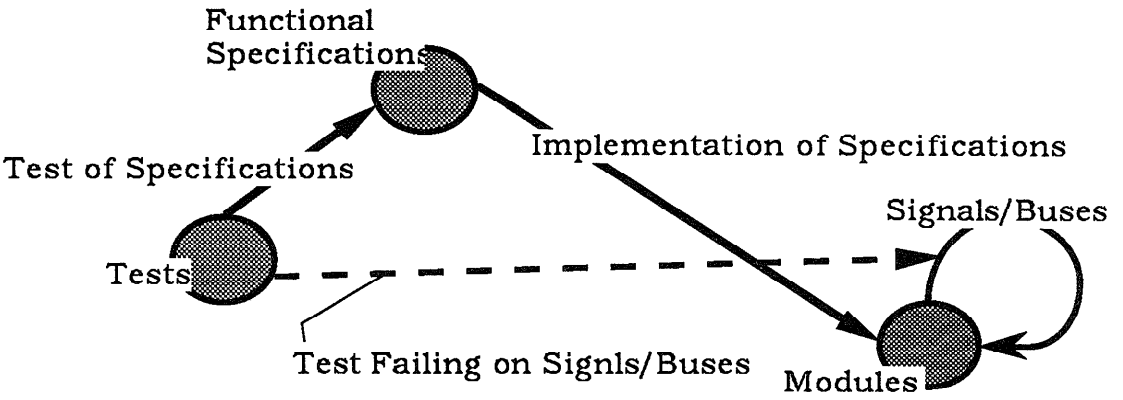

Figure 1. Search Space Schema for Focus of Attention

Similarly, failures of modules in workbook-level specification space are detectable through pathways of interaction between functional specifications and workbook-level modules. Single-stepping from failing tests to failing specifications to failing modules is an example of a weak search method for traversing along the pathways of interaction.

When local faults in one space cannot be mapped along pathways of interaction into local faults in another space, different pathways of interaction are required for successful diagnosis. *The suitable pathway should preserve the locality of faults.*

An instance of search space schema for focus of attention method for the interrupt controller (see Figure 2) demonstrates how the principle of locality is used to identify appropriate pathways of interaction. For example, test T3 verifies specifications C and E and test T8 verifies specifications A and C. Specification C is implemented by module *c*, specification E by modules *d* and *e*, and specification A by module *a*. If tests T3 and T8 both fail at the same time, specifications C and/or E are failing at the same time as specifications A and/or C are failing. Assuming singular faults and disjoint modules, one of modules *c* or *d* or *e* is failing at the same time as one of the modules *a* or *c* is failing. Thus, module *c* is failing and specification C is failing.

On the other hand, test T7 verifies specification B which is implemented by module *b*. If both T7 and T3 fail, *b* is failing at the same time as *c* or *d* or *e* is failing, which is not possible under the assumptions (singular faults and disjoint modules). Fault locality is not preserved along the pathways.

Different pathways of interaction that map test spaces into workbook-level spaces can be constructed by identifying signals (on which tests fail) with modules that generate these signals. In workbook-level spaces, for example, failure of tests T7 or T3 can be due to any of the signals *I, rd, halt, stf, mask, SEQ, pend0* or *SAV* failing. In test spaces, T7 failure is detected on *rd* and T3 on *halt*. In workbook-level spaces *rd* is generated by *d* or *e* and *halt* is generated by *d* or e, thus either *d* or *e*, for example, is faulty.

An application of the path following method to the above search space results in a computational solution that is based on the following heuristics:

(i) Start with a set of failing tests.
(ii) Use interactions of tests and functional specifications to propose a set of failing high level functional specifications, given a set of failing tests.
(iii) Use interactions of specifications via aggregation and decomposition to propose a set of failing low-level functional specifications.
(iv) Use an interaction across representations of low-level functional specifications and high-level workbook modules to propose a set of failing high-level workbook modules.

and

(i) Start with a set of failing tests.
(ii) Use the mapping between interactions of tests and functional specifications they check, and among modules via signals/busses to propose a set of failing high-level workbook modules, given a set of failing tests.

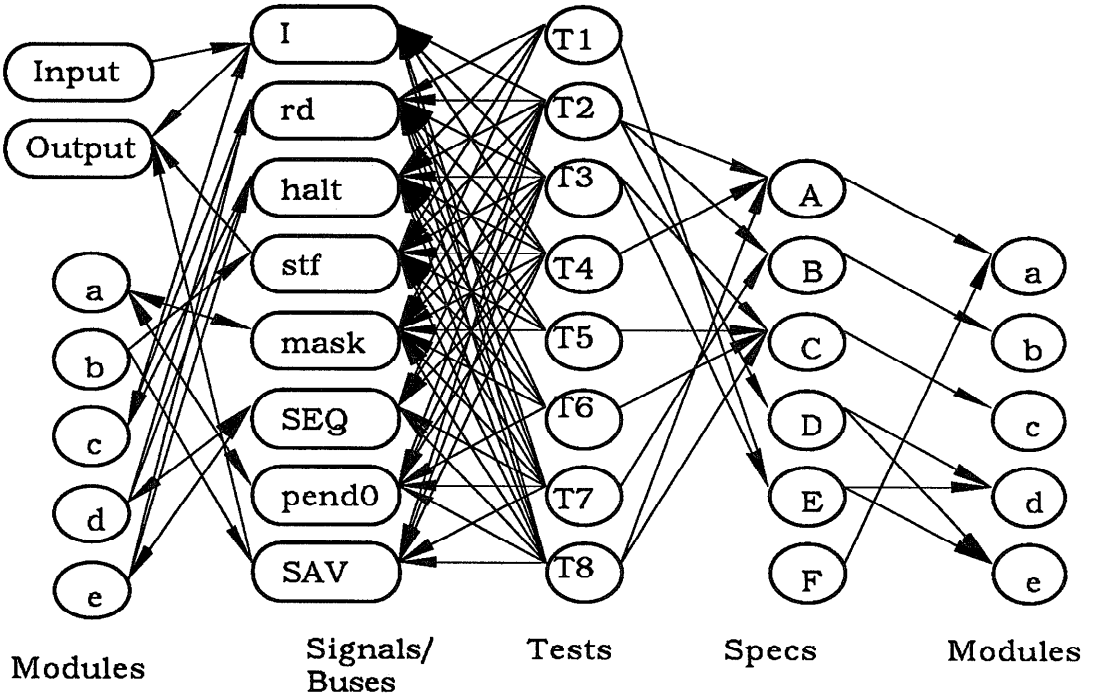

Figure 2: Instance of Search Space Schema for Focus of Attention

The above heuristics can be implemented by the following two rules:

IF
a substantial number of tests fail and a substantial number of test failures are caused by a variety of different test signals
THEN
among all the test that fail and that point to a possible failure of a set of functional units (areas) on the board, chose an area or areas that the majority of tests point to
ACTION
the areas most tests point to are likely to contain a fault.

IF
a substantial number of tests fail and a substantial number of test failures are caused by a small number of different test signals
THEN
among all the functional units or areas on the board that generate this small set of failing signals, chose the area that generates the most such failing signals
ACTION
the areas that generate the most failing signals are likely to contain the fault and the signals are good starting test points

Although in this paper we discuss only the *initial focus of attention* method, the corresponding search spaces for other domain dependent weak troubleshooting methods is given in Appendix A.

3.2 Evaluation of the Computational Solution

The validation of the computational solution was carried out as summarized in Figure 3.

A simulation model of the board as designed by the designers was constructed using a commercially available simulation environment running on IBM-PS/2 Model 70. A variety of faults such as bridge-faults, stuck-at faults and general component failure were introduced in the simulation. Bridge faults arise from shorts (see [Davis 84] for a description). The stuck-at faults consist of input or output of a component held to a constant value. Component failures were simulated by either eliminating the component or substituting one component for another. For example, a failed NOT gate behaves like a wire.

For each fault introduced, tests created by the designer were run on the simulation to produce test results for that fault. Test results for each fault were used as the input to the computational solution that generated the predictions about the area on the board where the fault exists (focus of attention).

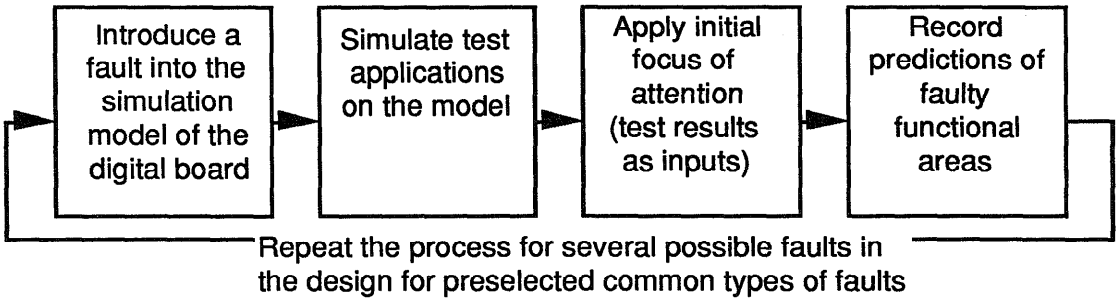

Figure 3. Process used to Evaluate the Computational Solution

3.3 Results

The results from testing the computational solution were evaluated by using the following two criteria:

$$Effectiveness = \frac{Number\ of\ Correctly\ Predicted\ Faults}{Total\ Number\ of\ Faults\ introduced}$$

$$Power = \frac{Number\ of\ Components\ in\ the\ Predicted\ Sub\text{-}set}{Total\ Number\ of\ Components}$$

Table 1. gives the distribution of the various types of faults introduced on the board.

Of the 27 faults introduced, the overall effectiveness was 77.8 %. The effectiveness measure of computational solution demonstrates that generated troubleshooting heuristics correctly predict a functional area on the board where the fault is located for 77.8% of all faults that were introduced.

The power of the focus of attention method is given in Table 2.

The reduction in search space measure from Table 2. demonstrates that the focus of attention method (on average) reduces the search area on the board where the fault is located to 27.9% of all the components (area) of the board. Comparing the best, worst and average cases in Table 2 demonstrates that computational solution might be imprecise, reducing the search by only to 70.5%, or very accurate, reducing the search to 8.8% of all components.

4. Conclusions

We are attempting to construct solutions to an interesting class of computational problems that impose unique constraints on an admissible solution. In this paper we have proposed a computational theory that exploits designer's knowledge obtained from the process of designing a device, and combines it with domain specific weak-troubleshooting methods by using the principle of locality as the organizational principle. We have also presented results on the feasibility of the computational theory by applying it to the process of designing digital devices to generate focus of attentions heuristics. The initial results, although limited, seem promising. We are in the process of extending the approach to incorporate additional weak methods.

Fault Type	Number of Faults Introduced	Number of Faults Caught
Stuck-at	10	7
Gate Failure	10	8
Bridge Faults	7	6

Table 1. Distribution of Faults Introduced to the Simulation Model

Power	Reduction in Search Space
Worst	70.5%
Best	8.8%
Average	27.9%

Table 2. Power of Focus of Attention Method

Bibliography

[Davis 84] R. Davis, "Diagnostic Reasoning Based on Structure and Behavior," Artificial Intelligence, No. 24, Volumes 1-3, 1984, pp. 347-410.

[Freiling 85] M. Freiling, J. Alexander, S. Messick, S. Rehfuss, and S. Shulman, "Starting a Knowledge Engineering Project: A Step-by-step Approach," The AI Magazine, No. 6, Vol. 3, 1985, pp. 150-164.

[Genesereth 84] M. R. Genesereth, "The Use of Design Descriptions in Automated Diagnosis," Artificial Intelligence, No. 24, 1984, pp. 411-436.

[Johnson 89] P. E. Johnson, D. Volovik, I. A. Zualkernan and C. E. Matthews, "Design Knowledge for Discovering Troubleshooting Heuristics," in Proceedings of IASTED Symposium on Expert Systems Theory and Applications, June 26-28, 1989, Zurich, pp. 17-21.

[de Kleer 87] J. de Kleer and B. C. Williams, "Diagnosing Multiple Faults," Artificial Intelligence, No. 32, 1987, pp. 97-130.

[Reed 88] N. E. Reed, E. R. Stuck, and J. B. Moen, "Specialized Strategies: An Alternative to First Principles in Diagnostic Problem Solving," Proceedings of the Seventh National Conference on Artificial Intelligence, Vol. 1, August 1988, pp. 364-368.

[Reiter 87] R. Reiter, "A Theory of Diagnosis from First Principles," Artificial Intelligence, No. 32, 1987, pp. 57-95.

Endnote #1.

The typical tests written by designers are used by designers for validation and hence are not sufficient for troubleshooting as they only check top level functional specifications.

Appendix A

Domain Dependent Weak methods	Pathways of interactions that define the relevant Search Space	Source of interaction knowledge (from designers doing a debugging task)
Path-following by Easter-egging	(i) Correct interaction of a part with its surrounding at the observation time	(i) Schematic diagrams (e.g. including ground, 'high' and clock signals)
Path-following by single-stepping	(i) Correct interaction of a part with its surrounding at the observation time (ii) Violated high-level interaction to track forward or backward	(i) Block-diagrams showing adjacent block interactions (ii) Block interfaces on the schematic diagrams
Path-following by subdivision	(i) Correct interaction of a part with its surrounding at the observation time (ii) Violated high-level interaction pathway to subdivide (iii) Correct interaction at a remote point along the interaction pathway	(i) Block-diagram showing block interactions and interaction pathways (ii) Block interfaces and interaction pathways on the schematics (iii) Test definitions (sources of stimuli for the initial correct and incorrect interactions along interaction pathways)
Compare-and-conquer	(i) Strong similarity of interactions of two or more parts (e.g. two signals are always 'high' at the same time)	(i) System and sub-system functional level specifications
Stateless analysis	(i) Stable states where the failure occurs (ii) Correct interaction of a part with its surrounding at the time of the failure	(i) System and sub-system functional level specifications (ii) Finite state diagrams (ii) State-control signals on the schematic diagram
Floating	(i) Pathway of interactions forming a feedback loop (ii) Appropriate signal value to force onto the floated part interface	(ii) Finite state diagrams (ii) State-control signal loops on the schematic diagram
Focus-of-attention	(i) Test-to-specification map (ii) Specification-to-subsystem map	(i) Test definitions (ii) System and sub-system functional level specifications (iii) Block-diagrams