HEWLETT PACKARD

An Expert System to Perform Functional Diagnosis of a Bus Subsystem

Chris Preist; Daryl Allerd; Ajay Gupta
Information Management Laboratory
HP Laboratories Bristol
HPL-91-16
January, 1991

diagnosis; expert system; manufacturing; electronics

Agatha is an expert system to help technicians test and diagnose failures in Hewlett-Packard's PA-RISC processor boards after their assembly. This paper focuses on the cachebus slice, the module in Agatha responsible for diagnosing faults in a bus subsystem on the board.

The slice uses casual knowledge, and is able to handle multiple and intermittent faults. By separating test specific knowledge from board specific knowledge, it can be easily updated to handle new board types. It has been extensively tested, and is in use at three sites within Hewlett-Packard. It is currently able to handle three different board types.

1 The Agatha System

Agatha is an expert system to help technicians test and diagnose failures in Hewlett-Packard's PA-RISC processor boards after their assembly. It is integrated with the PRISM functional test system [Schuchard,Kohlhardt], a Hewlett-Packard proprietary system which interfaces with the processor board to be tested and runs a series of functional test programs. Agatha, together with the PRISM system, provides a user friendly environment for board test and diagnosis. It consists of several smaller co-operating expert systems, called 'slices', capable of diagnosing different parts of a circuit board. It has been successfully deployed at three sites within Hewlett-Packard. In this paper, we focus on the slice responsible for diagnosing the cachebus subsystem, outlining its design and performance.

1.1 The board test process

The board test process consists of two phases; In-circuit testing, followed by functional testing.

In-circuit testing is performed by the HP 3065/3070 board tester. It consists of testing the components and connections on the board individually. It catches misloaded or missing parts, and most of the shorts and opens.

Functional testing involves testing the behaviour of various subsystems on the board, and verifying that they are behaving as they should. During functional testing, the PRISM tester runs a suite of tests to verify the behaviour of different subsystems on the board. When diagnosis is necessary, the technician takes the output from these tests and uses it to work out possible causes of failure. To gain further evidence for suspect causes, he/she can run further tests on the PRISM tester, or can carry out manual tests, such as probing parts of the circuit.

Boards can also be returned for diagnosis from any point in the production line after test, or from the field. In such cases, diagnosis is normally done at a PRISM board-test station.

Before the introduction of Agatha, the diagnostic process was not straightforward, for the following reasons;

- The PRISM tester is designed primarily for test and diagnosis at the level of integrated circuits. The data it produces for board diagnosis is of a low level nature, and can be overwhelming in quantity. This means it is difficult and time-consuming for the technician to deal with, and important information can be missed among the large volume of less relevant data.

- The repair recommended by the technician is not always correct. Several may be attempted before the board is fixed.
- As this testing has taken place immediately after manufacturing, it is unrealistic to assume that there can only be a single cause of failure. Hence the technician must be able to deal with a limited class of multiple fault possibilities.
- As manufacturing of PA-RISC boards spreads throughout the world, so does the need for the diagnostic process. This means that technicians with little or no experience of this task will need to perform it.
- The PA-RISC board family is continually updated to take advantage of the latest technology. This means that a new board is released on a regular basis, with basically the same test strategy, but with different specific test details, and a different structure.

To overcome these problems, the Agatha expert system was developed. It provides an easy-to-use interface with the PRISM tester and the diagnostic data produced in the test process. It can make diagnoses rapidly and accurately, and can be used by a technician inexperienced in PA-RISC diagnosis. It runs further tests automatically, as necessary, and guides technicians through any necessary manual testing if they so desire. As the knowledge specific to the PRISM tester and the knowledge specific to the board are kept separate, it is easy to update the system to deal with new board releases.

1.2 The Project

The Agatha project was a collaboration between Hewlett-Packard Labs, UK, (HP Labs) and Hewlett-Packard Integrated Circuit Business Division (ICBD). The collaboration was designed to be of mutual benefit to both organisations involved; ICBD was able to develop an expert system to aid technicians, and also became a centre of expert system excellence, allowing it to tackle future projects. HP Labs was able to gain further experience of technology transfer, and develop a firm understanding of specific problems to guide a longer term research program in model-based diagnosis.

The project consisted of several distinct development phases, with different responsibilities within each clearly partitioned between HP Labs and ICBD.

Knowledge Acquisition/Feasibility Prototype

Initially, we had to gain knowledge of the task performed, and produce a rapid prototype to give us some confidence in it. Knowledge acquisition was started by HPLabs, but devolved to ICBD staff as they gained experience of the techniques used. As this was going on, the feasibility prototype was developed to perform bus diagnosis only. It was written in PROLOG, and intended as a

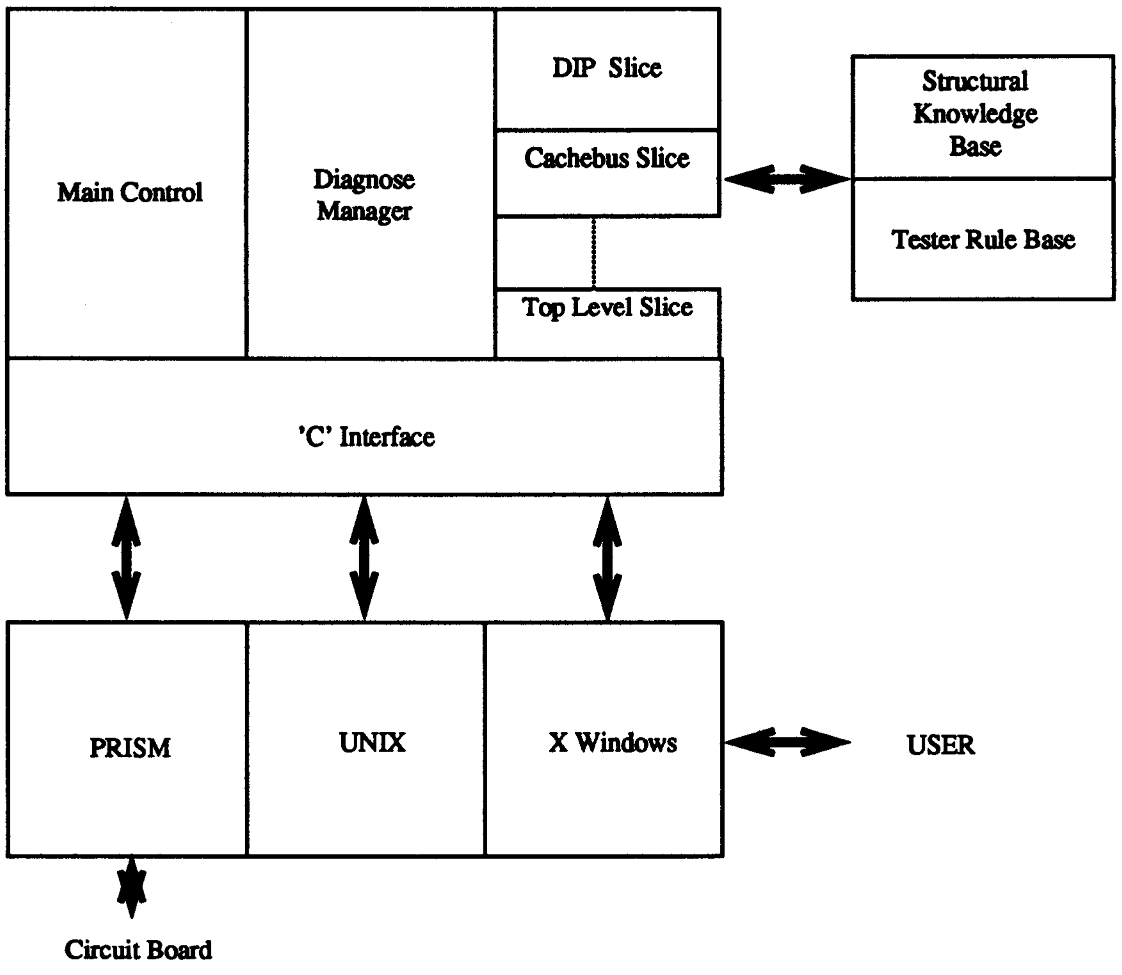

Figure 1 - Agatha Design

'throwaway'. The knowledge and algorithms it used formed a basis for the final cachebus slice, but were significantly adapted and entirely reimplemented. The prototype was not interfaced directly with the tester.

Proto I

For the first major prototype, it was decided to focus on three specific slices, including the cachebus.

During this phase, responsibility for the design and development of the expert system slices rested with HP Labs. ICBD was kept informed of all decisions, and was responsible for critiquing the evolving slice designs. ICBD was also responsible for all non-expert system parts of the system, in particular the interfaces with the PRISM tester and the user. Test cases were used to check the functionality of the system (See section 4).

Proto II

On completion of Proto I, primary responsibility for the entire system passed to ICBD. With consultancy from HP Labs where necessary, ICBD developed further expert system slices, and enhanced the original three. The system was released for in-situ testing to three HP manufacturing sites as trial customers.

Refine

On the strength of customer feedback, Agatha was refined to meet the customer's requirements more fully. This was done entirely by ICBD. When it was felt that the system was adequately stable, it was 'manufacturing released', making it available to other organisations within HP.

1.3 The Overall Design

As different parts of the diagnosis problem were found to require different inference techniques, we decided to subdivide the problem, and write different systems, the 'slices', to tackle each part. Hence Agatha is a suite of co-operating expert systems, each able to tackle a different aspect of the problem. To give the flexibility required to do this, we opted for Quintus PROLOG as an implementation vehicle.

These slices are integrated by another expert system, the 'Diagnose Manager', which is responsible for passing control from one slice to another, depending on previous test results and diagnostic hypotheses.

The slices and diagnose manager must communicate with the user, UNIX*, and the PRISM tester. This is done via 'C' programs. The user interface was developed in X windows.

The design is summarised in figure 1.

Further details about the project and overall design are given in [Allred, Allred2]. In this paper, we focus on the cachebus slice.

2 Diagnosing the Cachebus subsystem

2.1 The Cachebus Subsystem

The cachebus subsystem on the processor board consists of several large VLSI chips which communicate via a bus of around 150 lines. Each chip is connected to a subset of these lines, and is able to transmit onto the bus by driving values onto the lines. This can in turn be read off by the other chips.

The bus is pre-charged, so floats to high. When a chip drives a binary number onto the bus, it does so by actively pulling certain lines to low (to give zeros), and leaving others at high (to give ones).

At either end of each line in the bus is a termination resistor, which connects the line to a voltage source. Physically, these are clustered together into resistor packs, each holding about 6 resistors.

* UNIX is a registered trademark of UNIX System Laboratories Inc in the USA and other countries.

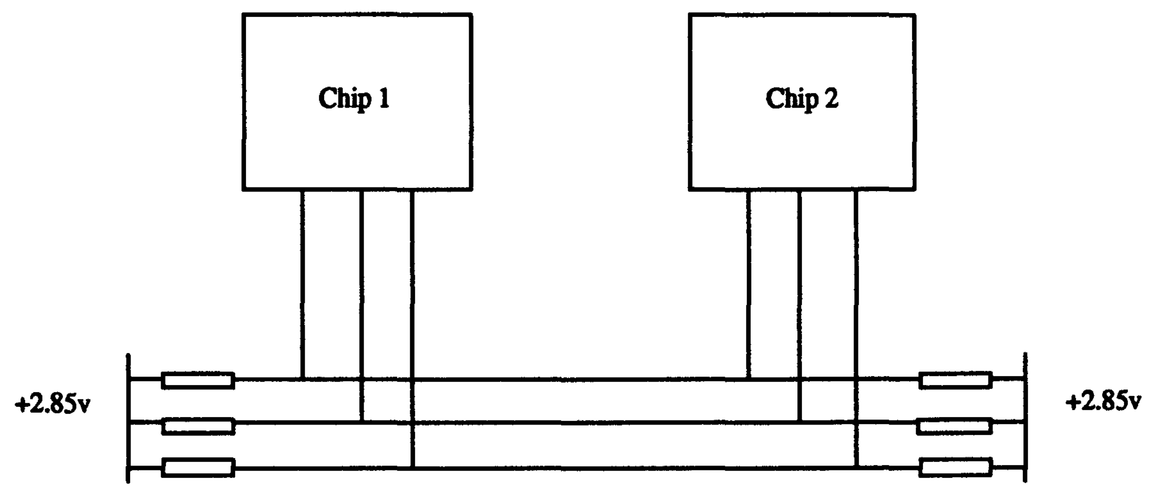

Figure 2 - A Simplified Version of the Cachebus Subsystem

A simplified version of this system is shown in figure 2.

2.2 Possible failures within the subsystem

Chip Failure

The chips have been tested prior to board assembly, but it is possible that they are damaged during the manufacturing process. Faults within chips manifest themselves, within the cachebus test, in the following ways;

Bad Driver - A pin on the chip connected to a given line in the cachebus is unable to drive the line as it should. It may be unable to drive at all, it may only drive some of the time, or it may drive the line even when it is not supposed to.

Bad Receiver - A pin on the chip connected to a given line on the cachebus is unable to receive data off the line properly. It may permanently receive a low value, permanently receive a high value, or may intermittently receive a high value when it should have received low.

Line Failure

Faults on cachebus lines are of the following nature;

Line Open - A break somewhere on the line, which results in the inability to communicate across it. When a chip actively pulls the line to low, only the other chips on the near side of the open will receive it properly. The others will receive it as high.

Open Pin on Chip - A break between a line and a chip. This stops any communication between the line and the chip, but will not affect any other communications.

Line Short to High - When a line is shorted to an active high, it will be pulled permanently up. This means any attempts to transmit low values along it will fail.

Line Short to Low - In this case, the line is permanently pulled down to low. Hence it will be received as a 0, irrespective of what is transmitted on it.

Two Cachebus Lines Shorted - When two cachebus lines are shorted together, they act as a 'wired and'. If either of them has a low transmitted on it, the other will be pulled down, and so will also be received low.

Resistor Pack Failure

When a resistor pack fails, it manifests itself as faults on several of the lines it is connected to.

Multiple Faults

Due to the nature of the manufacturing process, certain kinds of multiple fault are likely, and so should be considered as candidates. These include;

Bad chip - When a chip fails, it is common for several of the pins on it to fail at once.

Bad driver/receiver - When a pin on a chip fails, it may fail both as a driver and a receiver simultaneously.

Multiple short - A solder splash on the board would result in several adjacent lines being shorted together.

Intermittent Faults

Most classes of faults described above can be 'hard' or 'intermittent'. Hard faults always show up, whereas intermittent faults may show sometimes and pass other times. An example would be a poorly soldered junction, with a high resistance. Sometimes, a signal can pass through it, while at other times it will be blocked.

2.3 The Cachebus Test

The cachebus test consists of a sequence of mini tests. In each mini test, one of the chips broadcasts a binary number onto the cachebus, and all the chips (including the broadcaster) read the number back off. Differences between the broadcasted number and received numbers are output for diagnosis purposes.

For each chip driving, this is repeated with a 'walking 0' and 'walking 1' sequence across the cachebus. The 'walking 0' sequence consists of the chip driving each line in turn to low, and leaving all the rest high. The 'walking 1' sequence is the converse; each line in turn is left high, while all the rest are pulled low.

When the test fails, the technician uses the failing test patterns for diagnosis. Each failing test pattern consists of the driving chip, the receiving chip, the test pattern that was driven, and the bad received pattern. For a typical failing test, there will be around 1000 such patterns. Of all the tests carried out by the PRISM tester, the cachebus test is the one which produces the most data on failure. Using this data, the technician must propose possible diagnoses. This can be difficult and time-consuming, particularly when the technician has little experience of this diagnostic task. It is the automation of this task that we will discuss in the next section.

3 Design of the Cachebus slice

3.1 Design Decisions

3.1.1 Major Factors Influencing the Design

- The input received by the cachebus expert system is in the form of a very large number of observations of the circuit behaviour.
- Multiple faults on the cachebus are not sufficiently rare as to be negligible. Often, these will be 'single cause', such as the failure of a pin which is used as both a driver and a receiver. However, they can also appear at quite different places on the bus. For example, if several drops of solder splash on the bus, it could result in several line shorts.
- Intermittent faults must be handled.
- Certain 'catastrophic' faults, such as a chip which randomly drives even when it is not asked to, must be spotted.
- The system must be at least as fast and as accurate as an expert when diagnosing faults on the cachebus.
- The system must be easily updatable to cope with new boards in the PA-RISC family.
- While initially the responsibility of the corporate labs, the system must be maintained and extended by the division which is to have final responsibility for it.

3.1.2 Possible AI technologies

Heuristic Rule-based systems

A heuristic system, using a symptom/fault relationship, has often proved to be a valuable and efficient way of solving a diagnostic problem (for example, [Braunwalder]). However, they have not been particularly successful in handling multiple and intermittent faults. Often they are unable to. When they do deal with them, it can result in an explosion in the size of the rulebase. Secondly, the muddling of the structural and diagnostic knowledge in a heuristic system means it would be harder to update it to deal with new board releases. In addition, in our trials with the expert, it was observed that he employed predominantly causal knowledge to make predictions from his current hypothesis.

Model-Based Diagnosis Technology

Model based diagnosis technology overcomes the problems associated with heuristic expert systems. It provides a clean separation between the circuit structure, the behaviour of the different components, and the diagnostic process, so is easily updatable to deal with new releases. Also, it is able to handle multiple faults. However, at the time the decision was made, model based diagnosis was not a mature technology. There were some concerns over its efficiency. Also, little work had been done on processing large numbers of separate observations efficiently in a model based way. Finally, faults resulting in a change in the circuit structure, such as shorts between lines, had not been dealt with in a principled way.

A Causal Rule-based approach

As model-based technology was not sufficiently mature, but heuristic rules would be inadequate for the problem, we opted instead for an approach half way between the two. Rules are used, but the knowledge behind them is causal rather than heuristic. Structural knowledge referenced by the rules is kept in a separate knowledge base, allowing the rules to deal with observations on an abstract bus system. The rules are used to eliminate impossible or highly unlikely hypotheses. Heuristic knowledge is not discarded entirely; it is used to spot catastrophic failures, which could not otherwise be incorporated into the rules. It is also used to order according to importance the hypotheses the cachebus slice outputs (See 3.2).

3.1.3 Delivery Environment

The primary choice for the delivery environment was between a classical expert system shell, or a PROLOG system. We opted to deliver the system using Quintus PROLOG. The reasons for this decision are issues pertaining to the whole of the Agatha system, and not specifically the cachebus slice, so are covered in detail in [Allred2].

3.2 The Design

The overall design of the system is shown in figure 3. Raw data from the tester is parsed and preprocessed by a 'C' program. This is then put into a more abstract form by the expert system, and split into 'batches' according to the line in the bus which exhibits faulty behaviour. Hypotheses are then generated for each batch, and the observations in the batch are used to refute as many of these as possible. When this has been repeated for all batches, hypotheses are 'linked' between batches, to give single hypotheses capable of explaining observations on several lines. Finally, the hypotheses are ordered according to importance, by using heuristic rules.

The structural knowledge referenced by the rules is held in a separate database. It deals with the connections between chips and lines, the physical adjacency of lines on the board, and the correspondance between bits in the test patterns and lines on the board.

Observation Abstraction

Initially, the raw data output by the tester consists of a series of binary numbers, which are each transmitted by one chip, and received incorrectly by another. The first stage of abstraction, performed by 'C' routines, is to find the position of the specific bits which are received incorrectly by each chip. This information, together with the transmitter and transmitted binary number, is

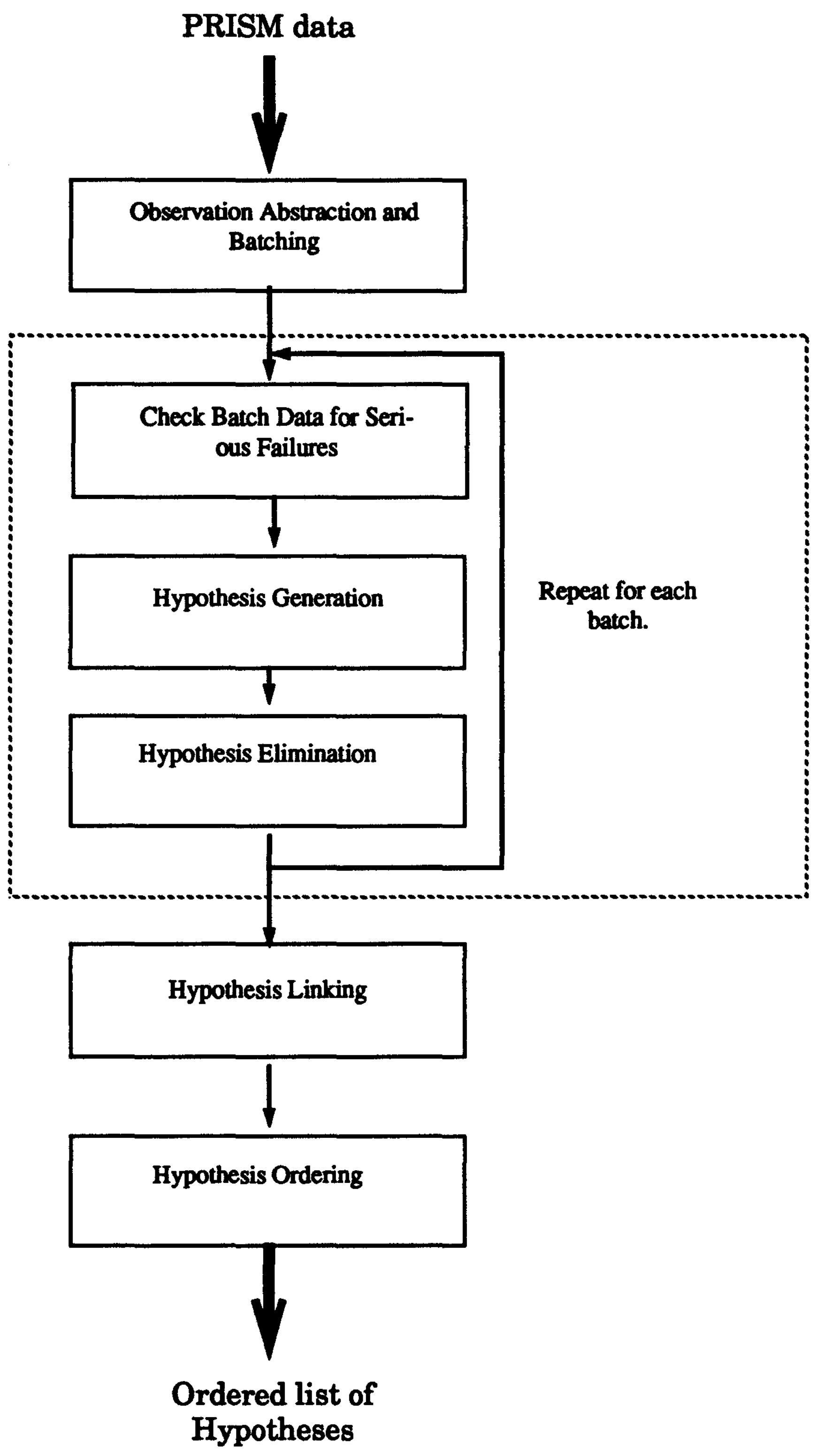

Figure 3 - Design of the Cachebus Slice

converted into a PROLOG readable form, and output. Observations which are received correctly are not represented in either the raw data or the abstract form.

From this, and the structural knowledge specific to the board type being diagnosed, the system is able to determine which lines are improperly received by which chip. Each value received incorrectly gives a 'pattern';

```
pattern( Driving_Chip, Driven_binary_number, Receiving_Chip,
            Bad_Received_Line, Bad_Received_Bit).
```

When a receiving chip receives a given binary number incorrectly on several different lines, it gives rise to several patterns.

Observation Batching

The process of batching is central to the handling of multiple faults by the system. Rather than considering the system as a whole, it is split up into many semi-autonomous subsystems, namely each individual line on the bus. This allows us to make the 'single fault per line' assumption, rather than the 'single fault' assumption (Though, as is explained later, we also consider certain multiple faults on a single line).

To be able to deal with each line independently, it is necessary to split the observations up according to which line they are relevant to, namely the line which is received badly. Hence the patterns are sorted into different lists, according to the 'Bad_Received_Line' field. These lists are referred to as 'batches', and are referenced according to the line with which they deal.

In the implementation, the process of abstraction and batching takes place concurrently.

Find Serious Failures

This rulebase is used to spot serious faults with behaviours so extreme that they invalidate the assumptions behind the rules used elsewhere in the system. It uses a heuristic match to filter these out. This provides the ability to spot the 'catastrophic' faults mentioned in section 3.1.1. Currently it only contains one rule, which detects a chip driving when not requested to. The rules used elsewhere in the system implicitly assume that there is no such chip.

Hypothesis Generation

Hypothesis generation takes place in a simple way, using almost no knowledge. For each batch of results, all hypotheses considered by the system are asserted as possible causes of the observed bad values. This is done by taking a template of hypotheses, and instantiating the information specific to the batch. This information consists of the line name, and the driving and receiving chips involved in the bad patterns. The instantiated hypotheses provide a list of hypotheses, each of which may explain the results in the batch.

Hypothesis Elimination

After generating all hypotheses for the observed behaviour on a given line, the system attempts to eliminate as many as possible, by firing the elimination rules. By using the observations and the structural knowledge, these eliminate those hypotheses which contradict with the observed behaviour on the line. These are derived by taking a causal relationship given by the expert, and converting it into it's contrapositive.

Hence, if Hypothesis1 => Observation1, we use the rule

not Observation1 => not Hypothesis1 to eliminate the hypothesis if the observation it predicts is contradicted by what is actually observed.

A simple example of this would be :

IF '1' is observed on the line at some time that '0' was expected

ELIMINATE bridge to ground and bridge to a cachebus line

BECAUSE these faults can only pull a line actively down.

Each rule has a 'because' field associated with it. These are not used in the inference process, but aid maintainers in understanding the system's behaviour.

Some rules require the use of 'pass' information. In these cases, any test which has no associated failure pattern on a given line is assumed to have been received correctly off this line.

The majority of knowledge is coded in the form of elimination rules. These are fired, in turn, for each batch of results.

Hypothesis Linking

Up to this stage, each line has been considered as an independent subsystem. However, lines are not completely independent. Some faults manifest themselves across lines. For example, the failure of a resistor pack, a bridge between cachebus lines, or a chip with several faulty drivers. It is the job of the linking rules to handle these.

After elimination, each batch has a small set of hypotheses which remain associated with it. The linking rules attempt to combine several hypotheses in different batches together to form a single hypothesis capable of explaining the observed behaviour of several lines. They also access the structural knowledge base when necessary. For example:

LINK : Bridge to cachebus on Line1 WITH: Bridge to Cachebus on Line2

IF: Line1 and Line2 are adjacent somewhere on the board

TO GIVE HYPOTHESIS: bridge between Line1 and Line2

Hypothesis Ordering

After linking, the remaining hypotheses are ordered using heuristic rules. These rules reflect the symptom coverage each fault provides and the relative likelihood of the different faults. The rules are, in order of priority;

1. Favour hypotheses which explain the largest number of batches.

2. Favour single-fault hypotheses over multiple fault hypotheses.
3. Favour non-intermittent hypotheses over intermittent hypotheses.
4. When none of the above apply, order the hypotheses according to the a priori relative likelihood of the two faults.

The final output of the cachebus slice is thus an ordered list of the hypotheses which remain for each given fault. This entire list is presented to the user, via the Agatha user interface, and the user is left with the decision of which of these to pursue first. The user will not necessarily opt for the one that Agatha has suggested is most likely, as others lower on the list may be far easier to test for.

Initially, each batch has 14 hypotheses (including intermittent and multiple faults) associated with it. On output, the system has 3-4 hypotheses remaining in total. From this, the technician can usually easily identify the problem.

3.3 Handling multiple faults

The Agatha Cachebus slice is able to deal with several different classes of multiple fault hypotheses;

Faults on Several Lines

The most common multiple fault during manufacturing is the appearance of faults on several different lines of the cachebus. Such cases are handled through the process of batching of observations according to line, and making the 'single fault per line' assumption in the reasoning process.

Independent Faults on the same line

Two faults are 'independent' if the behaviour of one cannot mask the behaviour of the other. For example, if two driving chips are 'stuck at 1', and unable to drive a low value when requested. As they will never be requested to drive at the same time during the test process, their faults will manifest in different observations, and not mask each other. Hence they are independent faults.

This is a rather less common occurrence than faults on different lines, and would not be handled by the single fault per line assumption. Where they are to be considered, they must be explicitly entered by the hypothesis generation process for a batch. For example, if a line batch contains observations of bad received values by both Chip1 and Chip2, it would be pointless to assert 'bad receiver on Chip1' as a possible hypothesis. It would be unable to explain all the observations in the batch. Hence it is necessary to assert the multiple fault hypothesis, 'bad receivers on Chip1 and Chip2', instead.

The rulebase does not need to handle such multiple fault hypotheses explicitly. The work is done by the inference engine; When one of the component faults of a multiple fault hypothesis is eliminated, the entire hypothesis is considered to be eliminated.

Dependent Faults on the Same Line

Two faults are 'dependent' if one is able to mask the behaviour of the other. For example a chip with a driver stuck high, and another chip with a receiver stuck low. When the driver is asked to drive low, it fails to do so. But the second chip receives the line low anyway, so the test passes, despite the existence of the faults.

Dependent multiple faults cannot be handled using the method described in the previous section. As they mask each other, the 'good' observations will eliminate them as possible hypotheses. Instead, they must be explicitly generated, and eliminated by the rule base. In other words, they are treated as separate single faults, and so require additional knowledge acquisition.

Currently, the only dependent multiple fault represented in the system is a bad driver/receiver pin on one chip. Because this fault has a single cause (namely a fault on that particular chip), it is significantly more likely to occur than other dependent multiple faults, and so must be dealt with.

3.4 Explanation

A disadvantage of a rulebase based on the elimination of hypotheses, rather than confirmation, is that it is harder to provide meaningful explanation facilities. It is difficult to answer questions such as 'why do you believe this?', as it is simply because it hasn't been ruled out as a possibility.

We had initially planned to provide an explanation facility that would be of use to the user of the system. However, user feedback in the initial stages suggested that this was a less important consideration than we had first thought. This was backed up by the experience at our alpha and beta test sites. However, we did still feel that it was necessary to provide some form of explanation for the use of the maintainer.

As it was to be used by the maintainer, rather than the user, it was acceptable (indeed, preferable), to allow it to include some 'inference' information. Hence we opted for a very simple approach; a trace of the firing of the high level rules. As the rules were fired, their result was written to a file; whether they failed or passed and what their result was on the list of possible hypotheses. The lower level rules behind these, such as the data and structure abstraction predicates, were not output. During the design stage, we decided that any more sophisticated explanation facility could take this file as an input, and provide a neater interface to it. Currently, we have not found a need for this.

4 Testing of the Cachebus Slice

Initial testing of the cachebus slice in the Proto 1 system was performed by using a suite of example outputs from the PRISM tester, each generated from a known fault. These were prepared using a 'poison board'. This is a board which can have faults of different types artificially induced on it. For example, a short could be induced by connecting a wire between two lines on the cachebus. The suite was designed to span all the different classes of fault the slice covers, including multiple and intermittent faults. Similar suites were developed for the other slices in Proto 1.

When the system was stable, it was released to customers as an alpha version, and statistics about its performance were gathered. These were used to further debug the system, where required.

5 Updating the Cachebus Slice

To ease the problem of updating, the cachebus slice was designed with a clear distinction between the tester-specific rulebase and the board-specific structural knowledge base. This means that when a new board is released which is tested with the PRISM tester, only the structural knowledge base needs to be changed.

This updating process is automated. 'C' routines access design data files which are used by the PRISM tester, extract the structural knowledge required by the cachebus slice, and convert it into PROLOG clauses. The only intervention needed by the maintainer is to make minor alterations to these routines, if the design data files change format. The test-specific rules do not need to be altered.

The cachebus slice has been successfully updated to deal with 3 different board families.

6 Benefits of the Cachebus Slice

As the Cachebus Slice is part of the Agatha system, it is difficult to provide independent data about it's specific benefits. The benefits of the entire system are discussed in [Allred2]. However, it is possible to point to certain advantages provided by the cachebus system specifically.

- It provides faster diagnosis of cachebus tests. Agatha takes between 1 second and 3.5 minutes to perform diagnosis of the cachebus, averaging around 30 seconds. The expert averages around 4 minutes to perform the same task.
- It has improved the accuracy of cachebus diagnosis. There is no data for the Cachebus slice independent of the rest of the system. However, customer feedback confirms that diagnosis of faults in the cachebus has indeed been improved.
- Easily updatable to deal with new board families.
- As a part of the AGATHA expert system, it can now be easily deployed at new sites, and can be used by technicians not used to dealing with these board types.
- It has provided a good impetus for research into model based diagnosis at HP Labs. [Eshghi, Preist]]

7 Conclusions

While heuristic expert systems can be too simple to tackle a diagnostic task properly, and model-based approaches are not able to cope with certain complexities, a causal rule-based approach can yield good results. They can provide a system able to deal with multiple and intermittent faults, and allow separation of different types of knowledge for easy upgrading and maintenance.

By developing the system as a joint project between HP Labs and the final system owners (ICBD), we were able to transfer the expertise effectively from one party to the other. This freed the labs from any long-term maintenance commitment, and established a centre of excellence in expert system technology at ICBD. It also gave HP Labs experience of real problems, thus allowing them to focus their longer term research program in model-based diagnosis on areas which would be of particular use to HP divisions.

Acnowledgements

Thanks to Rick Butler, our expert, and Caroline Knight, for help with knowledge acquisition.

References

[Schuchard] Schuchard and Weiss. 'Scan Path Testing of a Multichip Computer' in ISCC'87 Digest

[Kohlhardt] Kohlhardt, Gaddis, Halperin, Undy and Schuchard, 'Design, Verification and Test Methodology for a VLSI Chip Set' in HP Journal, Sept '87

[deKleer] deKleer,J and Williams,B 'Diagnosing Multiple Faults' in Artificial Intelligence 32

[Preist] Preist,C and Welham,R 'Modelling Bridge Faults for Diagnosis in Electronic Circuits' in Proc. 1st International Workshop on Principles of Diagnosis

[Eshghi] Eshghi,K 'Diagnoses as Stable Models' in Proc. 1st International Workshop on Principles of Diagnosis

[Allred] Allred, 'The Agatha Project - The Development of a Diagnostic Expert System. 'HP ICBD Internal Report

[Allred2] Allred, Lichenstein, Preist, Gupta 'Agatha, An Integrated Expert System to Test and Diagnose Complex PC Boards' Submitted to Innovative Applications of Artificial Intelligence, 1991

[Braunwalder] Braunwalder, K and Zaba,S. 'RBEST: an expert system for disk failure diagnosis during manufacturing' in Practical Experience in Building Expert Systems, ed Bramer, M. John Wiley 1990.