The Induction of Fault Diagnosis Systems from Qualitative Models*

D.A. Pearce
The Turing Institute
36 North Hanover St, Glasgow, Scotland

Abstract

This paper describes a methodology for the automatic construction of diagnostic expert systems, and its application for fault diagnosis of a satellite's electrical power subsystem.

The synthesised knowledge base is compared with an existing expert system for the same application built using a commercial expert system shell. Both systems have been tested using a real-time satellite simulator which has the capability to fail components.

A traditional knowledge-engineering approach involves building a prototype which is refined until satisfactory results are obtained. This process is error-ridden, as even in small systems, rules can conflict, be irrelevant, or missing. It is never clear when a system is complete and validation is always difficult.

As an alternative, a fault diagnostic knowledge base can be automatically synthesised from a qualitative model of the device. This is achieved by systematically simulating all component failures. Individual failures are used as examples. A learning algorithm is applied to the examples to output a set of diagnostic rules. The resulting rules are complete and consistent with the qualitative model and diagnose component failures in the model 100% accurately. Validation becomes a higher level problem of ensuring that the qualitative simulation accurately models physical device behaviour.

1 Introduction

The traditional knowledge engineering approach to building an application in the fault diagnosis field, is to chose a suitable expert system development tool and manually enter rules which cover all the possibilities the domain specialist can envisage. Even after a number of iterations the system is unlikely to be complete and consistent. Validation of the knowledge base can be a major problem if the system is to be used in a live environment.

For some systems an alternative approach is to use a qualitative model of a physical device and its possible failure modes as a specification. It is then possible to generate a knowledge base which accurately diagnoses failure of any single component in the model. The problem of validation becomes the higher level issue of ensuring that the qualitative simulation accurately models the behaviour of the physical device. We have found that this is a task the domain specialist can accurately perform.

We have developed a methodology for building fault diagnosis expert systems, and have tested it in an aerospace application. The application has driven development of software tools capable of being adapted to other problem domains.

The application chosen was fault diagnosis of the electrical power subsystem of an on-station satellite. A real-time numerical simulator for the satellite had previously been developed, and used for operator training. This allowed testing of the knowledge base on real data. In addition, a rule-based expert system using the commercial shell Envisage [Systems Designers, 1986] had previously been developed to analyse the simulator output for faults. Thus a direct comparison could be made with a separate expert system designed to perform the same task, but constructed via the traditional knowledge engineering methodology.

It is our intention to build on the existing tools to develop a general purpose software environment, capable of automatic generation of diagnostic knowledge bases.

2 Qualitative Modelling

Simulation of the satellite power subsystem has been achieved through constructing an executable model [Mozetic *et al.*, 1988]. The model is *deep* as regards the distinction between *deep causal* knowledge and *shallow, operational* knowledge. We define shallow-level knowledge as knowledge that is sufficient for performing the task itself, but typically without any representation of the underlying causal mechanisms. Deep knowledge, on the other hand, captures an underlying causal structure and facilitates reasoning from first principles. When running the application, explanations and advice can be derived from the underlying model. Take, as an example, failure of any relay switch in the network of switches used to route solar array power to charge the batteries. This will not only be correctly detected, but advice is given in the form of an alternative (minimum change) relay configuration that will restore battery charge. This information is taken straight from the model's representation.

The model is *qualitative* in the sense that it does not deal with electrical components represented numerically as voltages and currents over time, but with components represented by symbolic descriptions that specify qualitative features. For example, in the model a voltage level may be considered low, normal or high. Such a qualitative modelling approach has several advantages over conventional

*Work has been carried out in collaboration with British Aerospace with funding from the European Space Agency numerical modelling:

- The qualitative view is closer to the domain specialists descriptions of the reasoning about the operation of the device.
- To execute the model we do not have to know exact numerical values of the parameters in the model.
- The qualitative simulation is computationally less complex that the numerical simulation.
- The qualitative simulation can be used as a basis for constructing understandable explanations.

The model is used for automatic synthesis (through simulation) of a shallow, operational representation of the knowledge. This knowledge is bulky and is compressed using an inductive learning algorithm. Figure 1 illustrates the various levels of knowledge and transformations between their representations.

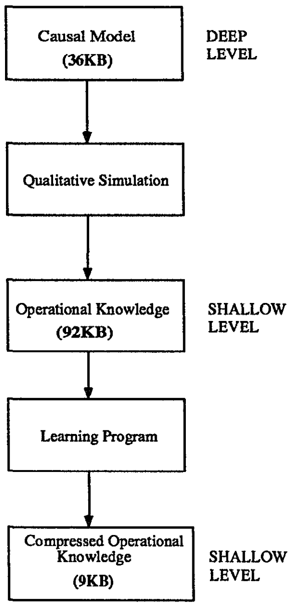

Figure 1: Deep and shallow levels of knowledge

3 Synthesising the Knowledge Base

The task of generating a knowledge base is split into the following sub-tasks:

- Development of a static, qualitative component model
- Development of heuristic knowledge about behaviour
- The simulation of all possible component failures to generate a set of examples
- Compression of the examples to generate a diagnostic rule base

The first two items jointly constitute the qualitative simulation. This model can be tested against the behaviour of the physical device by simulating failure of components and observing results. Satisfactory model behaviour can be established before compression takes place.

3.1 The qualitative component model

The component model defines all the basic components, their initial states and their relationship with each other. Indicators used in the model are also defined stating under what circumstances they change value.

In our application, the model of the power subsystem is based on Figure 2.

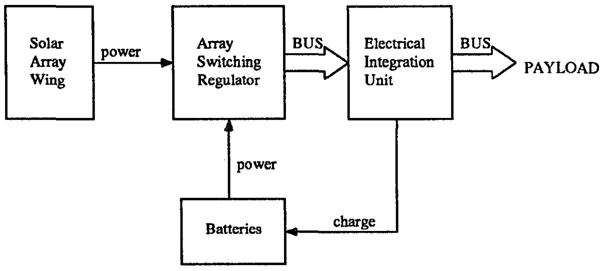

Figure 2: Block Diagram of Satellite Power Subsystem

Power from the solar arrays is routed through the Array Switching Regulator (ASR) to supply power to the bus when the spacecraft is in sunlight. During eclipse the power comes from two batteries. The ASR contains a set of switches used to enable or disable solar array sections. During operation, a comparator detects a rise or fall in the bus voltage and automatically opens or closes ASR switches to restore the bus voltage to normal. The Electrical Integration Unit contains relay switches necessary for main charging or trickle charging the two batteries. The Prolog model contains a description of each component, and its relationship with upstream and downstream directly connected components. This is expressed using the *comp* predicate. The arguments are used to specify name, type, controlling device, input connections and output connections.

```
comp(array_3A,a,[],[],
     [or_switch_2, asr_switch_2]).
comp(comparator,c,[],
     [bus],[asr_switch2,...,asr_switch10]).
```

These two *comp* clauses are used to define components *array section 3A* and the *comparator*. Here, array section 3A has no downstream (input) connections, and is directly connected upstream (output) to both ASR switch 2 and override switch 2. The comparator has direct input from the bus, and its output is connected to all the ASR switches.

In order to start the model, initial states for each device must be specified. This is achieved using the *init_state* predicate. The initial state may be conditional, depending on the state of some external factor such as the mission phase or the payload connected to the bus.

```
init_state(array_3A,no_power) :-
        init_state(mission_phase,eclipse).
init_state(array_3A,power) :-
        init_state(mission_phase,post_eclipse);
        init_state(mission_phase,solstice).

init_state(comparator,s000001111) :-
        init_state(load,3).
```

3.2 The heuristic behaviour knowledge

Behaviour rules are used to specify the operation of the static component model. A set of behaviour rules exist for each component in the model, indicating how a change of state affects its neighbouring components.

Simulation of faults in the power subsystem can be achieved through changing the state of the component to be failed, then firing the behaviour rules repeatedly until the system reaches a new stable state.

The inference mechanism used is form a queue of active components, and to fire a behaviour rule for the component at the head of the queue. Behaviour rules are tested until one is found which can successfully fire. This is a deterministic operation of finding the first rule that can fire from a linear search of the rule set. The result of a rule firing will typically be to change the state of another component, which then joins the active component queue. The behaviour rules are repeatedly fired for the component at the head of the queue until the queue empties and the model reaches a static state.

This complete process for any one component in the satellite model is typically executed in about a second.

As a result of components changing state, visible indicators may also change. Indicators are observable values which show the status of the system at various points. In the satellite application, indicators are the telemetry values which the spacecraft sends to earth at regular intervals. Following a simulated failure, a generated example makes use of the indicators as attributes, with the component failure as the *class* or *decision*. Two of the behaviour rules from the satellite application are listed:

```
rule 1 ::
if   component bus of_type b is low
and  component comparator of_type c is AnyState
then comparator takes_next_up_state.

rule 9 ::
if   component comparator of_type c is
s000111111
and  component asr_switch_3 of_type s is open
then asr_switch_3 becomes closed.
```

3.3 Generation of the examples

To generate a set of failure examples, each component in the model is failed under every possible combination of external factors on the model. This gives a complete example set covering all possibilities. The example set is reduced by applying some common sense constraints. An example is not considered if it is a duplicate of an existing one, or if an example describing a failure is identical to a normal state of operation with respect to the observable indicators. The second constraint is necessary to prevent suggestions that faults have occurred under normal operation. For example, failing a solar array during an eclipse will go undetected until the eclipse is over. It would be unhelpful for the system to suggest that during normal operation any one of the solar arrays could have failed!

In the satellite example, 61 functional components were failed under 8 payload values 0 through 7, and 3 solar phases (eclipse, post eclipse and solstice). This resulted in 1464 (61x24) examples, which reduced to 708 under the above constraints.

3.4 Rule Induction

A version of the AQ rule induction algorithm called AQR has been used [Michalski and Larson, 1983; Clark and Niblett, 1987]. AQR induces a single decision rule for each failure in turn. It is able to deal with conflicting examples by giving a rule with a disjunction of failures. Two examples conflict if they contain identical attribute values, but have a different decision. This can happen in our application when identical telemetry indicator values are seen after failing different components.

Generally the process of rule induction takes a set of incomplete examples as input, and forms a general rule which covers the example set. This can best be described as *induction*. However, in the modelling case, the examples are complete and consistent and we generate performance rules by *data compression*. It the satellite application some 75 diagnostic rules are produced from over 700 examples, with an average of only 3 attribute tests to identify a fault, where each example contained over 30 attributes.

4 Existing software tools

The software has been developed on a Sun 3 workstation using Quintus Prolog. Use has been made of *Paneltools II* [Hoff, 1987] an internally developed Prolog graphics package which provides an interface to Sun windows.

Three separate processes have been developed:

- A non-interactive analysis program which systematically fails every component in the model for every external factor. The output of this program is a file of examples.
- Rule induction on the set of examples using a version of the AQ algorithm. Again this is not interactive, and its output is a file of diagnostic rules.
- A graphical, interactive diagnostic program which uses the output from the induction process to diagnose faults introduced in the model. This program combines simulation of component failure with expert system diagnosis using the knowledge base.

The user interface consists of a graphical qualitative simulation of the satellite power subsystem in which components such as switches and solar array panels are seen to change state. Using a mouse the user of the system can fail components, view the graphical simulation and receive advice from the diagnostic knowledge base, which detects invalid patterns in the status indicators. The user may, in response to a piece of advice, perform the action such as closing a switch to verify the expert system's advice is correct.

5 Comparison with the hand-crafted system

The two knowledge based systems, both designed to detect the same set of component failures, are compared in two different ways. Firstly, the real-time satellite simulator was used to create telemetry data files covering a range of different failure situations. Each KB was then run using this data as input. Secondly, the internal integrity of each KB is checked using the Knowledge Integrity Checker (KIC) [Pearce, 1987], a tool previously developed at the Turing Institute.

The Envisage system contains roughly 110 rules split fairly evenly between forward chaining, and backward chaining rules. The forward chaining rules are used to control the execution from one KB area to another, while the backward chaining rules are used to assign values to attributes. Total development time was of the order of 6 man months.

The rule induced KB contains some 75 rules which forward chain from telemetry indicator values to reach a diagnosis. This is the executable set of rules used for comparison. However, for maintainability, the behaviour rules in the model are used. There are 64 high-level behaviour rules used to describe the operation of the power subsystem. Total development time for the modelling approach was of the order of 3 to 4 man months.

5.1 Testing KB on simulator data

At present the satellite has not been launched. Telemetry data is not available therefore the simulator has been used. When the induced rules were tested on a small (14) set of failure situations created using the real-time satellite simulator, a 100% success rate was achieved in correctly diagnosing the faulty component.

This compares favourably with the manually constructed expert system which managed to detect 10 out of the 14 as failures, a 72% success rate.

Although the sample size is too small to give accurate success rates, it is evident that the manually constructed rules contain omissions, which are not present in the automatically synthesised rule base.

5.2 Checking KB rule integrity

Confidence in the internal integrity of a knowledge based system can be increased through the use of the Knowledge Integrity Checker (KIC) which takes the knowledge base as an input, and produces as output a set of rules which have been identified as possible errors. The system indicates possible omissions, contradictions and errors, giving the system designer a powerful facility to help development and debugging of a knowledge based application.

The software has been developed in Quintus Prolog on a Sun workstation. KIC performs validation on knowledge bases represented in a Horn clause format, through detecting logical, structural and semantic inconsistencies in the rule base. The following inconsistencies are capable of being detected:

- Unreachable clauses
- Dead-end clauses
- Cyclic clauses
- Type checking
- Incompleteness
- Subsumption

Both knowledge bases required conversion into a Horn clause format before they could be suitable for input to the KIC. This was straightforward in both cases. The rule induced knowledge base was automatically converted to change the form of each rule, to which manually constructed type information was added. The other knowledge base, developed using the *Envisage* shell from Systems Designers PLC., had to be manually converted. Again, type information was added. Although Envisage supports a fairly rich environment for developing applications, the underlying logic of the rules mapped easily onto Horn clause logic.

Running the induced knowledge base through the integrity checker showed up no errors or inconsistencies. On the other hand, when the Envisage knowledge base was checked, the following problems were noted:

1. A type error which would prevent one rule from ever successfully firing.
2. Two unreachable clauses, which could be removed.
3. Four dead-end clauses, which indicated the definition of a particular predicate was incomplete.

Figure 3 summarises the results:

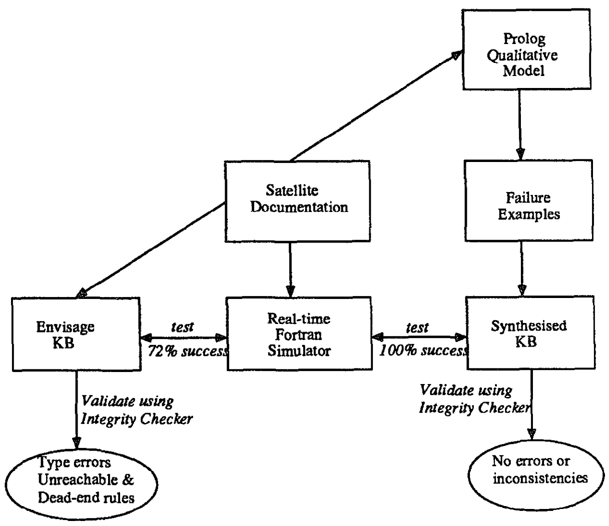
Prolog Qualitative Model
Satellite Documentation
Failure Examples
Envisage KB
test
72% success
Real-time Fortran Simulator
test
100% success
Synthesised KB
Validate using Integrity Checker
Validate using Integrity Checker
Type errors Unreachable & Dead-end rules
No errors or inconsistencies

Figure 3: Comparison of results

6 Conclusion of the study

For fault diagnostic applications, the study has shown that qualitative modelling techniques can be used to substantial benefit.

- Synthesised diagnostic rules are 100% accurate with respect to the model.

- Qualitative models are easily constructed using a logic programming representation, and are extensible.
- Applications can be constructed more cheaply, requiring less of the domain specialists valuable time.
- Explanations during execution are improved through the ability to refer to the underlying structure of the device.
- Validation is easy for the domain specialist to perform, especially if the model simulation is interfaced to a graphical display.

7 Future Plans

The current plan is to develop and build on the techniques used to date to produce a general purpose modelling environment for constructing diagnosis expert system applications. Research into current developments in qualitative modelling will be pursued to ensure the process of mapping the real world to a model can be achieved with elegance and efficiency.

7.1 Production of a general purpose environment

Development of the modelling environment will be driven by one or more significant application. Careful choice of application will ensure that the modelling environment will support similar types of application in a different problem domain. One of the chosen applications will be in the fault diagnosis field, although it is hoped other types of application can be identified which will fit the model. For example, design or planning problems may be possible to be tackled using the environment.

7.2 Multiple faults

The present modelling software assumes single failure operation, rather than combinations of failures. In the satellite application this is sufficient, as the hardware is designed with single-fault tolerance and all operations are built around this premise. It is therefore reasonable that single component failure diagnosis is performed.

However, for many other diagnostic applications, single failure cannot be assumed. In these cases the model representation must be capable of handling double or multiple combinations of failures. It will be necessary to apply certain constraints on permissible multiple failures, otherwise exponential growth will occur in the systematic simulation and hence the example set. In [Mozetic *et al.*, 1988] a qualitative model of the heart is built to interpret ECG signals. Here multiple combinations of some 30 basic disorders are dealt with in the model. However, a maximum of seven of these were medically possible at one time, and further constraints such as disregarding logically and physiologically impossible combinations, and also medically uninteresting states further reduced the possibilities.

7.3 Hierarchical modelling

Hierarchical modelling could allow the division of a model into logically separate parts, with the ability to define subparts, and specify a relationship between the various levels and units on the same level.

This would allow large models to be more easily maintained, and would also facilitate a graphical representation of the model on the screen with an ability to expand and hide sub-modules.

7.4 Time based reasoning

Implementation of time-based reasoning can open up areas previously ill-suited to a qualitative modelling approach. The current system assumes that any component in the model will only change state as a direct result of some other component-based action. It could be possible to model time as a qualitative entity, and allow definitions of components against time. A timed-based simulation could be achieved by stepping through time units, where units are defined according to the particular application. In the satellite application, this would allow elegant modelling of such functions as a battery losing its charge during an eclipse etc. QSIM, the qualitative simulation developed by Kuipers [Kuipers, 1986] allows such time based simulation. In this system components may be defined as monotonically increasing, decreasing or remaining constant over time.

References

[Clark and Niblett, 1987] Peter Clark and Tim Niblett. Induction in noisy domains. In I. Bratko and N. Lavrac, editors, *Progress in Machine Learning (proceedings of the 2nd European Working Session on Learning)*, Sigma, Wilmslow, UK, 1987.

[Systems Designers, 1986] Systems Designers. *Envisage Reference Manual.* Technical Report, Systems Designers PLC, Camberly, Surrey, 1986.

[Hoff, 1987] A.A. Van Hoff. *Paneltools II Documentation and Reference Manual.* Technical Report, The Turing Institute, Glasgow, 1987.

[Kuipers, 1986] B. Kuipers. Qualitative simulation. *Artificial Intelligence*, 29(3):289–338, 1986.

[Michalski and Larson, 1983] R. S. Michalski and J. Larson. *Incremental generation of VL_1 hypotheses: the underlying methodology and the description of program AQ11.* Technical Report ISG 83-5, The University of of Illinois at Urbana-Champaign, Department of Computer Science, Urbana, 1983.

[Mozetic *et al.*, 1988] I. Mozetic, I. Bratko, and N. Lavrac. Automatic synthesis and compression of cardiological knowledge. *Machine Intelligence 11*, 1988.

[Mycroft and O'Keefe, 1984] A. Mycroft and R.A. O'Keefe. A polymorphic type system for PROLOG. *Artificial Intelligence*, 23:295–307, 1984.

[Pearce, 1987] D. Pearce. *KIC: A Knowledge Integrity Checker.* TIRM 87-025, The Turing Institute, Glasgow, 1987.