Diagnosing multiple faults using knowledge about malfunctioning behavior

Tim Hansen
Department of Computer and Information Science,
Linköping University, S-581 83 Linköping, Sweden

Abstract

A technical fault diagnosis system based on knowledge about the structure of a device and the behavior of its components is presented. This approach allows knowledge about components to be reused for new devices and simplifies maintenance of the fault diagnosis system. A new method for diagnosing multiple faults using this representation is presented and discussed. The method assumes that both normal functioning and malfunctioning behavior are described for components in the form of user-defined constraints. A prototype implementation for parts of the method exists and has been compared with a traditional fault-tree implementation of a trouble-shooting system for an separator application.

1. Introduction

During a joint project with Alfa-Laval[1] developing a fault diagnosis system for one of their separators we saw some potential advantages of moving from a traditional expert system approach into a more component-oriented one. In this paper we present a model where we represent the function of a technical device in a compositional way. That is, the function of the device can be derived from the components and the connections between the components. This way of representing knowledge is often referred to as *deep models*. In the paper we also present an implemented method of diagnosing faults in technical devices, which uses both the normal and malfunctioning behavior of the components in the device.

------

[1]Swedish manufacturer of separators.

To be able to compare the two types of diagnosis methods, we have used the same separator process in both systems. In a separator process, two mutually insoluble liquids with different densities are pumped into a rotating bowl. In this bowl the liquids are separated to different outlets. To control the process there is an electronic control unit, which receives input from sensors, and through valves and other components control the flow in the separator.

In the expert system developed by Alfa-Laval, the knowledge about the separator was represented as *symptoms*, *hypotheses* and *faults* [Nordin87]. Symptoms, hypotheses and faults together built a fault tree, emulating the expert's course of action during the diagnosis. This way of representing knowledge is often referred to as *shallow models*. The evaluation of the diagnosis system gave that it preformed well. However, we found some problems that the technique could not cope with. The problems are listed below:

Multiple faults are hard to detect using shallow models. This can be done much easier with deep modeling as described later in this paper. The reason why multiple faults are hard to diagnose using shallow models is that the faults are related to symptoms and/or absence of symptoms. If there are two faults present in a device, it is not always the case that the union of the symptoms of the two faults will be the actual symptoms of the device. It could happen that the union of the two faults has the same symptoms as another single fault.

Maintainability of the knowledge in a shallow model is difficult to obtain. This is due to the fact that a fault in a part of the device can depend on symptoms observed in a number of other parts of the device. If the device is changed and a few symptoms are changed, then the result could be that many of the faults have to be changed.

Explanations in shallow models are often not related to the functioning of the system. A problem with text in shallow models is when changes are made in the rules, then the explanations may become inconsistent. In [Swartout83], a number of problems with such *canned text explanations* are described. By using the knowledge contained in the deep models, causal relations between components can be traced. Further explanations can be given to why a component is in a given state.

Reusability of knowledge in shallow models is often very low. If we, for instance, have a part of a device, and express the faults of that part in symptoms of the device, then the knowledge of the part cannot be used when the part is built into another device. The faults of the parts have to be expressed in symptoms of the new device, which can be totally different.

Development of shallow models often involves a single expert. The reason for this is that two or more experts often disagree about the best way to the diagnose faults. If a single expert is used, then this expert has to be consulted many times to develop a diagnosis system. Using deep models several experts could be interviewed concerning, developing the function of some component in the device.

To solve the problems mentioned above, we have developed a model and a method to diagnose faults in technical devices using deep modeling. In the model the knowledge about a device is represented as the functioning of its components and the connections between the components. Here the components functioning includes both the normal functioning and malfunctioning of the component.

To derive the functioning of the device from the components, we have developed a procedure which we call ***bidirectional simulation.*** The bidirectional simulation detects all valid fault combinations of the device. To reduce the number of interesting combinations we start the bidirectional simulation with the restriction that only one component can be malfunctioning, later two or more components may be assumed to be out of order.

In this paper the proposed model is found in section 2, the bidirectional simulation is presented in section 2.5. In section 3 a procedure for reducing the search needed for bidirectional simulation and heuristics for finding faults are discussed. Section 4 contains a discussion about the contribution of the work and some problems with the model.

2. The model

The model we have built, represents the functioning of the device in a compositional way. By this we mean that the function of the device can be derived from the functioning of the components and the connections between them. In the description of the model below, the connections between the components are called *signals.*

In the model, the functioning is represented using *qualitative values.* For qualitative values, the value space is expressed using a limited number of values, and often the derivatives of the values are expressed too. The main reason for using qualitative values in the model is the need for reasoning with unknown values. This can be accomplished by trying each of the possible values, provided that there is only a small number of them.

In the literature there exist different types of qualitative values, such as the value space {-,0,+} used by de Kleer and Williams [Kleer84a] [Williams84]. In other works, Kuipers uses his notation with landmarks [Kuipers86]. Yet another type is reported by Forbus [Forbus84]. In the model presented here we do not define the value space, we just assume that it is limited.

2.1 Signals

As previously mentioned, the communication between the components is carried out by signals. Each signal can contain a number of different attributes, which reflect different types of flows between components. If we for example take a hose, then this hose could contain attributes such as flow, pressure, type-of-liquid, heat and so on. A flow here could mean that there is something coming through the hose, and the pressure could say if the pressure is high or not. As the system is dealing with qualitative reasoning, it supports attribute values of both the amplitude and of the derivative.

Signals do not have any direction attached to them. All the information about directions in the signals is determined by the components. This is done because, a component that is malfunctioning can change the direction of flow in the surrounding signals. Compared to the direction the flow would have had, if the component had been normally working. An example of this is the grounding error in an TTL-circuit. This problem is described in [Davis84].

2.2 The components

The components represent the lowest level of knowledge in the system. By this we mean that there is no more knowledge about physical laws in the system, than implicitly represented in the rules of the components.

In the components the knowledge is separated into two parts. The first one contains the normal working of the component, ok-states in example 1. The second part contains the malfunctioning of the component, which is further separated into different fault types. Faults belong to different fault types depending on how close they are to normal-functioning. The malfunctioning behavior for one fault is shown in example 1 under fault-states.

```
Ok-state
        Dis-ok 1     if    (AND  (EQ Inwater water ok)
                                 (EQ Bowl liquid ok))
                     then (SET Discharge liquid ok)
        Dis-ok 2     if    (AND  (OR (EQ Inwater water no)
                                      (EQ Inwater water little))
                                 (EQ Bowl liquid ok))
                     then (SET Discharge liquid no)
Fault-state
        Dis-fault 1  if    (AND  (OR (EQ Inwater water no)       possible
                                      (EQ Inwater water little))  fault
                                 (EQ Bowl liquid ok))
                     then (SET Discharge liquid no)
                     if    (AND  (EQ Inwater water ok)      fault
                                 (EQ Bowl liquid ok))       operation
                     then (SET Discharge liquid no)
                     cause       Separator paring disc fouling
                     symptoms
```

Example 1

The components can be seen either as a model of the real world, or as an conceptual model built by the knowledge engineer. The components are built to simulate the function they represent. The representation language contains operators such as IF, THEN, SET, EQ, AND and OR. For the operators SET and EQ the syntax is (SET <port> <type> <value>). Here the <port> and <type> are the port and the type of the component to be set. The <value> is the value, that the port is going to be set to. This <value> could also be an input/output port from which the value should be taken. The EQ operator compares the values instead of setting a value as done in SET. In this implementation the number of operators are kept down to reduce the design complexity.

2.2.1 Normal states

The normal states describe the functioning of the components, as they will function when no faults are present. The normal functioning is modeled in rules, where each rule contains an IF <premise> THEN <statement>. The <statement> of the rule describes what values the output signals should have, if the <premise> is true. An example of normal rules are dis-ok 1 and dis-ok 2 in example 1.

In the <premise> part of the rules the operators OR and AND are allowed arbitrary many times. For the <statement> part of the normal rule, the only allowed operators are SET and AND. The OR operator is not allowed here, because the rules are used to simulate the function of the component, and the component cannot have two responses to the the same input.

2.2.2 Fault states

In the fault state part of a component description there is a list of all known faults that can occur in the component. For each of the listed faults there is a functional description of the faults malfunction, similar to the function described for the normal state using IF <premise> THEN <statement> rules. However, there is a difference compared with the description of the normal states: In the fault states, the rules are separated into two parts. *Fault operation* and *possible fault.* The rules of the faults in *fault operation* describe the actual malfunctioning of the device. The *possible fault* is when there is a fault, but it is not possible to separate them from the normal state functioning of the component. If a component is operating in *possible fault*, the input values of the components has to be changed in order to verify the fault.

2.3 Simulation

From the description of a device given in the above model and the input values of the device, it is possible to qualitatively simulate the function of the device. In simulating the behavior of the device it is possible to let one or more of the components be malfunctioning. This will cause faults to appear in the simulation, which can be used to train trouble shooters on a conceptual level. Compare for instance the STEAMER project [Hollan84].

As the model presented uses qualitative values, not all the input signals of the device are needed to perform a simulation. If an input signals is missing, there is only a limited number of possible values it can have. For each of the values in the value space, the simulation will be continued using the selected value, and the result of the simulation will consist of a number of different solutions.

2.4 Constraints

The use of numeric constraints can be found in several works such as [Stallman77] and [Maleki87]. There are also examples of qualitative constraints such as the ones used by Kuipers [Kuipers86]. The constraints we use are qualitative and do not use predefined operators, instead we build the constraints from the simulation rules in the components. The components are divided into a normal state and a number of fault states, and for each of the states in the component a constraint is built.

A constraint for a state uses all the rules in that state. For each of the rules, the <premise> part is examined to see if it contains any OR operator. If it does, the <premise> is deduced to <premises>, not containing any OR operator. For each of the new rules created from the same rule, the <statement> part is the same. All the rules not containing the OR-operator are the constraints of that component state.

The constraints we get above are used to determine the values of the unknown input and output ports. The signals determined by the rules are propagated further to other components. The best case is when only a single rule matches the known input and/or output ports of the component. If, however more than one rule match the known ports, all rules are listed and the signal produced by the first rule is propagated further. The other rules in the list can be applied later if an inconsistence is detected, during propagation of the first one.

2.5 Bidirectional simulation

The bidirectional simulation takes one or more signal values of a device and puts them into the model of the device. All the values except one is given to the model and the constraint propagation is started for the given values. This propagation ends when no more values can be determined, without making a guess about a state in a component. The reason for leaving one signal value is that the propagation for that value is done when the bidirectional simulation is started.

Now the signal value that was not propagated above is selected, and the propagation is started with this value. The propagation is continued until a choice between two or more states in a component is encountered. At this point, the environment is saved together with the list of possible component states. If the constraints in the selected states have more than one possible outcome, then the alternatives are added to the list.

At this point the first choice on the list is selected and the propagation is started again. This process will go on until no more propagation is possible, or an inconsistence is detected. If no more propagation can be made, we say that a *device state* is reached. A device state consists of all the signal values and the component states of the device.

An inconsistence is found if either a value is given to the model and another is found by the bidirectional simulation, or the constraint rules cannot match the input and output port values. If a device state is found or a inconsistence is detected, then the backtrack is started. The system will backtrack until an untried component state are found. If there are no untried component states left, the bidirectional simulation will stop, and the output will be the possible device states.

The main difference between simulation and bidirectional simulation is that in simulation the rules are only used one way, and in bidirectional simulation the rules are used for constraint propagation. This difference causes that the simulation to only accept input to the device, while the bidirectional simulation can take any signal value as input.

Dependency directed backtracking as described in [Dole79] or in [Stallman77] would be suitable for the bidirectional simulation, although this is not yet implemented. Another interesting thing would be to implement the bidirectional simulation using an ATMS, as the one in [Kleer86]. In that paper de Kleer claims that this is the type of problem where the ATMS works best.

3. Finding the fault

3.1 Pruning the solution space

The pure bidirectional simulation for a device given so far, will in most cases give more than one device state. Many of the device states proposed by the bidirectional simulation will have a large number of faulty components. For most of these cases, it is not interesting with many faulty components, and a heuristic rule to reduce the simulation could be to first assume one fault. If needed later two or more faults can be assumed. There will not be any loss of generality to make the restriction of assuming one fault, since all of the components can be assumed to be faulty. This way of coming around the complexity is also done by Davis in [Davis84] and [Genesereth84].

The bidirectional simulation is started with the restriction that only a single component is allowed to be faulty. This will restrict the search needed to come up with the possible device states. The possible results of the bidirectional simulation, where only one faulty component is allowed are:

1 There are several device states with one fault, although more than a single component is faulty in the device states.
2 There are several device states with one fault, in all of the device states the same component that is faulty.
3 There is one device state with one fault.
4 There are no device states that have one fault.

As the model often gets few signal values to start with, case 1 is likely to appear. This means that before some of the device states can be ruled out, more information is needed. This will be described later in the section about finding the best question.

In case 2 and case 3 the fault is found, or at least a possible fault is found and the procedure is terminated. For case 4 the fault is not to be found with one fault, and the procedure has to restart with a two faults assumption. This is done by restarting the bidirectional simulation, but allowing two components to be faulty.

This process of starting with one fault and continuing with two faults could continue until all the components are found faulty. However, there is not much point in trying to diagnose more than a small number of faults. If almost all the components are broken then there is a serious fault, which this procedure is not designed to handle. Furthermore, if many of the components are faulty, then most of the function will be out of order, and the gains of using this procedure may be reduced.

In bidirectional simulation the ATMS was said to behave very well, but with problems of pruning the solution space with one fault, the ATMS would not behave very well according to de Kleer [Kleer86].

3.1.1 Finding the best question

If we have more than one device state that satisfy the constraint determined by the model and the signal values given to the system, the goal is to reduce the number of device states. This might be achieved by asking the user for more signal values. The best question is found by comparing the different device states, and by looking for a signal which values has been guessed by the bidirectional simulation, and if it can be determined from the real world, will eliminate many device states. In [Kleer87] de Kleer gives a probability-based approach of how to determine the most possible faulty component, and thus also the best question to ask.

One complication with this way of deciding which question to ask, is that not all signals are equally easy to measure. We suggest that the cost of measuring signals should be regarded.

3.2 Verifying the states

Not all faults are noticeable when the device is in a certain state. To verify these faults, the device has to be forced into another state, in order to detect the fault. An algorithm to carry this out this is the D-algorithm described in [Keravnou86]. The above fault states that cannot be detected, are found under possible faults in the model.

Some problems of changing a component state can be tested by heuristics, for instance if there is an electric valve and there is a suspicion that it cannot break. A heuristic rule within the component, would have to test if the valve could change from open to close.

3.3 Lack of knowledge in the model

It could happen that the pruning of the solution space described above fails, for instance if there was no fault in the system, or the functional descriptions of a fault were missing. Even though the system has no idea about where the fault is, it could try to guide the user to the faulty part of the device.

The first thing is to start the bidirectional simulation, allowing no components to be faulty. If a device state is found, then there is no fault in the system or the model is not precise enough to find any faults. However, if no device state is found, information about fault function in the component is missing. It could also be the case that there is a type of fault outside the components, such as the bridging error in [Davis84].

If the malfunctioning behavior description in the components is missing, we could try to locate the fault by using techniques such as the ones suggested in [Davis84] or [Kleer87]. Both of these methods use only normal behavior of components.

4. Discussion

All procedures presented in this paper, from section *the model* and the procedure up to and including the section *pruning the solution space* have been implemented on a XEROX 1186 Lisp-machine.

As earlier indicated we have compared the ALFA-LAVAL fault diagnosis system using a traditional expert system shell, with the model built in our system. From these two implementations we have been able to verify some of the statements done about shallow and deep models in the introduction section of this paper.

Multiple faults can be found using the normal and malfunctioning behavior of the components in the device, as have been seen *in pruning the solution space* earlier.

The possibility of maintaining the knowledge base has increased by only allowing knowledge about the device within the components. The work needed to perform updates is reduced by keeping the knowledge modular.

In the work no implementation has yet been made to see if the technique could provide better explanations. However, Steels claims that deep knowledge gives better explanations [Steels87].

The reusability of the information in the models is improved as the components can be reused in other devices without modifications.

We feel that the development of components in a device or parts of a device can be done by different experts. However, it is then important is that the experts agree on the connections between components.

4.1 Contributions

In the work presented in this paper, we have given the following contributions to the expert system technique:

The knowledge in the model presented is modular into the components of the device. This modularization makes the knowledge in the device more reusable and easier to maintain.

The use of knowledge both about normal functioning and malfunctioning can lower the number of measurements on the real world device in a diagnosis, compared with the number of measurements needed by the approaches without fault models e.g. Davis and de Kleer [Davis84] [Kleer87]. An example of this would be the two components in figure 1, there should be a flow through the two components, but there is not. In a diagnosis system using only knowledge about normal functioning of the devices, the only way to find the faulty component of the two would be to measure the flow between the components. However, by using a model with both knowledge about normal function and malfunction, the fault in this example could be found if it is only one of the components that can cause the flow to break.

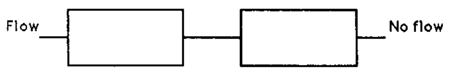

Figure 1

The model presented in this paper with its normal functioning and malfunctioning can be used as a ground for compiling deep models into more shallow ones, as proposed by Nordin and Milne [Nordin87] and [Milne87].

4.2 Problems with the model

All modeling of the real world have problems. Some problems are general for modeling, and other problems are related to the modeling technique used. An example of the latter is the limitations reported by Struss on qualitative reasoning [Struss87]. In the model we use, we have also found some problems, which are reported here.

There is a limitation in the model presented, due to the fact that only device states are used. We neither support modelling of changes of components, nor provide any means for reasoning about sequences of changes for a component. That is, in the bidirectional simulation the changes of a component is not recorded, although for a human trouble shooter, changes of components can give valuable information in detecting the fault.

Using the component-based technique presented here, there is also a problem with causal relations between components. This problem is partly due to our intention of not destroying the modularity of the components, as described in connection with *interdependent components* below.

4.3 Complete break-down

One problem with the approach presented in this paper is if the device, is not functioning at all. For example, if the power supply to an electric device breaks down, then the fault of the device can not be found using our approach.

A way to overcome a problem like that would be to combine shallow and deep models, as proposed in [Milne87].

4.4 Feedback

In the system presented in this paper there is no powerful way to handle feedback. The process of feedback is described in [Kleer84b].

With the system presented, the simulation part can use feedback. But the bidirectional simulation is not built to handle feedback. In the bidirectional simulation there is no algorithm to detect feedback, neither is there any algorithm for reasoning with feedback.

Algorithms to detect feedback are available. The odd loops detection by Goodwin [Goodwin87], will detect feedback if present. In [Kleer84b] de Kleer detects and reason about feedback in electronic circuits.

Although de Kleer [Kleer84b] reasons with feedback, it is not clear how this should be used to diagnose faults in our model.

4.5 Interdependent components

In figure 2 there are three components involved, a pump, a secure valve and a valve. These three components give rise to a problem for the bidirectional simulation. The problem is that none of the components knows anything about each other. This is due to the modularity of the components, each of the components represents just its own function. Here the pump causes a flow and a pressure, the secure valve opens if the pressure is very high and the valve switch if it is activated.

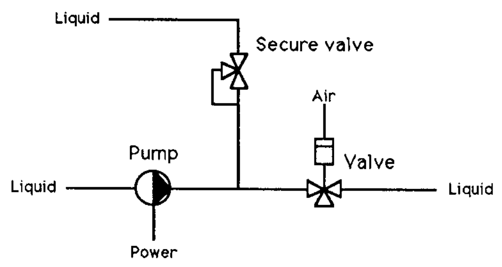

Figure 2

The functioning of the three components is: The secure valve will switch to open if the valve is closed and the pump is pumping. This functioning can not easily be modeled in the system by the bidirectional simulation. One way to solve this problem would be to allow negotiation between the components. However, if have parallel valves, or if there are several components between the pump and the valve in figure 3, then the negotiation is not well suited. An approach to solve the problem would be to take the component descriptions, and for each of the components add the causal relations to other components. One approach that could work to solve this problem is the algorithm for detecting strongly and loosely connected components in networks, described in [Goodwin87].

The model presented in this paper can handle the problem above by breaking the modularity principle. This is done in the example above by adding a causal relation from the secure valve to the valve.

5. Summary and Conclusion

Our work started from the experiences of developing and maintaining a traditional expert system for trouble shooting based on shallow-reasoning heuristics. The procedure for detecting multiple faults described in this paper grow out of a project, where we tried to develop a formalism for describing components of a device in a modular and reusable way.

The core ideas of our approach are based on using qualitative descriptions of components and their behavior in terms of normal functioning as well as malfunctioning. By bidirectional simulation one or more possible faults, which can explain an observed malfunction of the complete device, are identified.

Essential for the practical application of the method is that efficient heuristics for pruning the search space can be used. Practical experiments indicate that this is not an unrealistic assumption. However we also recognize that for practical purposes, deep models for describing the functioning of more complex devices will have to be combined with heuristics-based shallow-reasoning models. Still we believe that our method for diagnosing multiple faults contribute significantly to the possibilities of developing and maintaining trouble-shooting expert systems for new devices built from known components.

Acknowledgments

Thanks to Kristian Sandahl, Sture Hägglund and Henrik Nordin for useful discussions and insightful comments on this paper.

References

[Doyle79] Jon Doyle, A Truth Maintenance System, Artificial Intelligence 12 (1979), page 231-272.

[Davis84] Randall Davis, Diagnostic Reasoning Based on Structure and Behavior, Artificial Intelligence 24 (1984), page 347-410.

[Forbus84] Kenneth D. Forbus, Qualitative Process Theory, Artificial Intelligence 24 (1984), page 85-168.

[Genesereth84] Michael R. Genesereth, The Use of Design Descriptions in Automated Diagnosis, Artificial Intelligence 24 (1984), page 411-436.

[Goodwin87] James W. Goodwin, A Theory and System for Non-Monotonic Reasoning, Linköpings Studies in Science and Technology, Dissertations No. 165, Linköpings Universitetet.

[Hollan84] J. D. Hollan, E. L. Hutchins and L. Weitzman, STEAMER: An Interactive Inspectable Simulation-Based Training System, AI Magazine, vol 5, no 3, page 15-27, summer 1984.

[Keravanou86] E. T. Keravnou and L. Johnson, Competent Expert Systems: A case study in fault diagnosis, Kogan Page Ltd.

[Kleer84a] Johan de Kleer and John Seely Brown, A Qualitative Physics Based on Confluences, Artificial Intelligence 24 (1984), page 7-83.

[Kleer84b] Johan de Kleer, How Circuits Work, Artificial Intelligence 24 (1984), page 205-280.

[Kleer86] Johan de Kleer, Problem solving with the ATMS, Artificial Intelligence 28 (1886), page 197-224.

[Kleer87] Johan de Kleer and Brian C. Williams, Diagnosing Multiple Faults, Artificial Intelligence 32 (1987), page 97-130.

[Kuipers86] Benjamin Kuipers, Qualitative Simulation, Artificial Intelligence 29 (1986), page 289-338.

[Maleki87] Jalal Maleki, ICONStraint, A Dependency Directed Constraint Maintenance System, Linköpings Studies in Science and Technology, Thesis 71, LiU-Tek-Lic 1986:11, Linköping University.

[Milne87] Robert Milne, Strategies for Diagnosis, IEEE Transactions on systems, man, and cybernetics, vol. smc-17, No. 3, May/June 1987, page 333-339.

[Nordin87] Henrik Nordin, Reuse and Maintenance Techniques in Knowledge-Based Systems, LiTH-IDA-R-87-18, Linköping University, September 1987.

[Stallman77] Richard M. Stallman and Gerald J. Sussman, Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis, Artificial Intelligence 9 (1977), page 135-196.

[Steels87] Luc Steels, The deepening of Expert Systems, Artificial Intelligence Communications, vol. 0 No 1, august 87, page 9-16.

[Struss87] Peter Struss, Problem of Interval-Based Reasoning In Fruechtenicht et al. (eds), Wissensrepr–sentation und Schlussfolgerungsverfahren fuer teechnische Expertensysteme, Muenchen (1987) Also: Technical Report, SIEMENS INF 2 AMR-1-87.

[Swartout83] William R. Swartout, XPLAIN: A System for Creating and Explaining Expert Consulting Programs, Artificial Intelligence vol. 21, no. 3 (1983), page 285-325.

[Williams84] B. C. Williams, Qualitative analysis of MOS circuit, Artificial Intelligence 24 (1984), page 281-346.