FAILURE MODE ANALYSIS USING MULTILEVEL FLOW MODELS

Bengt Öhman*

* Department of Information Technology, Lund Institute of Technology, Box 118, 221 00 Lund, Sweden.
Fax: +46 46 222 4714, E-mail: bengt@it.lth.se

***Abstract*— Failure mode analysis for complex systems is an important, and sometimes challenging, task. In this paper, a method for failure mode analysis based on Multilevel Flow Models (MFM) is presented. When an MFM model of the system exists, it is possible to perform an automated failure mode analysis, and even to give failure predictions in an on-line system in real time. The system is implemented in a version of the MFM Toolbox, a program for editing and testing MFM models on a Windows-based PC.**

I. INTRODUCTION

When designing a complex industrial system, it is often important to study the effects of failures on other parts of the system. Traditionally, this has been done using manual methods, by filling out forms by hand, such as the Failure Mode and Effects Analysis method (FMEA). Some methods with automated computer support have evolved, such as the Fault Tree Analysis method (FTA), and the Goal Tree - Success Tree method (GTST). There are also tools available to automate parts of an FMEA analysis, see [10]. All these methods will collectively be called Failure Mode Analysis methods, or FMA methods for short, in this paper. An overview of some methods is available in [2]. GTST is described in [9], and FTA is described in [1].

A problem with many of these methods is that it is not possible to analyze several failures at the same time. Also, since the methods are mostly manual, it is possible to miss important scenarios during the analysis. Another important aspect is what will happen if the process is changed — how will the new system behave? Are the previously obtained results still valid for the new process? These questions can be difficult to answer, and they may require that the whole analysis be done all over again.

Model-based methods have an advantage in this scenario, especially with computer support. The part requiring most work — the construction of the model — is already done. What is needed is to change the part of the model that is affected by the restructuring of the process, after which all the original cases plus any new ones may be evaluated again. This will require less time and effort compared to a manual analysis, since the analysis itself is mostly automated.

What is described in this paper is a new method for failure mode analysis based on Multilevel Flow Models (MFM) [8]. MFM is a graphical modeling language for systems of flows of mass, energy, or information, and is a good base for several diagnosis algorithms, such as alarm analysis and measurement validation, see [3]. Thus, if an MFM model of the system exists, it can be used for several other diagnosis algorithms, thereby increasing the value of the model. The FMA method developed for MFM has many nice properties, such as the possibility for on-line prediction of failures in real time, and it would make a nice complement to the traditional FMA methods.

II. ABOUT FAILURE MODE ANALYSIS

The purpose of performing a failure mode analysis is to investigate the effects of one or several failures in a system. This can be done using several methods, among the most well-known are FMEA and FTA (see part I). The general idea with these methods is to find all possible failures of a system, and to relate these to each other so that the effects or causes of a fault can be determined. This can be done using a graph, such as the one in Figure 1. Here, the faults numbered 1 to 4 are called "basic faults," and the combination of one or several basic faults leads to larger faults. The analysis is then performed by starting with a large fault, and then investigating the tree to find the basic faults leading to this failure.

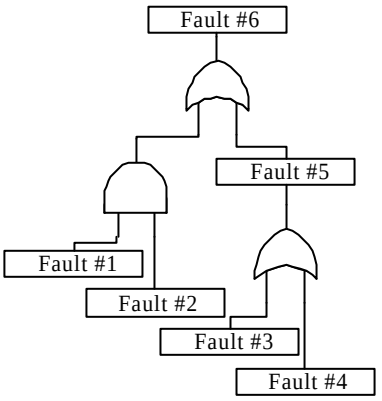

Figure 1. An example of a fault tree.

A shortcoming with the common methods for failure mode analysis is that the analysis is often done during the design phase for the system, and the results of the analysis are not easily retrievable by the operators during operation. It may also be the case that some of the failure modes are not analyzed, due to insufficient effort in the analysis. With these thoughts in mind, it is easy to see that an on-line, real

time FMA system would be advantageous. Such a system would be able to provide the operators with predictions of future failures based on the current state of the system. The FMA algorithm for MFM is capable of both off-line and on-line analysis, and could be used in both the design phase for a system, and when the system is running.

III. About MFM

Multilevel Flow Models were invented by professor Morten Lind [8], and they are primarily used for modeling goals and functions and their relations of technical systems. However, the MFM models make an excellent base for diagnostic reasoning, and several algorithms have been developed. MFM models describe the goals and functions of a system. The *goals* describe the purposes of running the system (the answer to the question "why?"), and the *functions* describe the capabilities of the system (the answer to the question "how?"). The functions are connected in terms of *flows* of mass, energy, or information. The flow structures are connected into *networks*, to which the goals are attached. The functions are realized by *components*, which are the physical objects in a system.

The connection between the networks and the goals are called *achieve relations*, because they indicate that goals are achieved when the flow functions in the connected networks are working. There are also *condition relations*, which tells that a certain goal has to be achieved for the conditioned function to be available.

Goals

The goals in an MFM model describe the intentional goals of running a system. Examples of goals are "keep the water level within an acceptable range," "produce electrical power," and "cool the pump." Without knowing the true goals of running a system, it is practically impossible to build a good MFM model of the system, since the whole purpose of the system is to fulfill the goals.

Components

The components represent the physical structures of a system. This can be, for example, a piece of pipe, a water tank, or an accumulator. The components are usually not shown in an MFM model.

Functions

The functions in MFM represent the capabilities of a system, such as "transport water," "supply electricity," or "prevent transport of radiation." The functions may or may not be associated with a physical component in the system. A component may also be associated with several functions. An electrical pump, for example, may have both the function of transporting water, and the function of acting as a sink for electrical energy. The MFM functions are these:

- A *source* is a function that is capable of providing an unlimited amount of mass or energy.
- A *sink* is a function that acts as a drain of mass or energy.
- A *transport* is a function that is capable of transporting mass, energy, or information.
- A *barrier* is a function that is capable of preventing the transport of mass, energy, or information.
- A *storage* is a function that is capable of storing mass or energy.
- A *balance* is a function that connects one or more inflows to one or more outflows.
- An *observer* is a function that translates sensor values to information.
- A *decision* is a function that represents control actions by humans or automatic controllers.
- An *actor* is a function that can transform information into physical actions.
- A *manager* is a function that is used to represent management of a system, for example PID-controllers, human operators, etc.

These functions are shown in Figure 2.

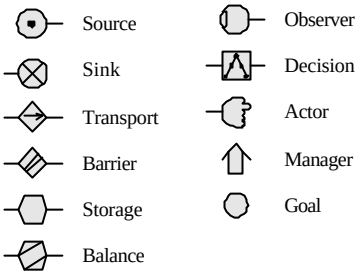

Figure 2. The symbols that represent MFM functions.

Relations

A set of connected flow functions is called a *network*. To the networks, goals can be connected via *achieve relations*. When a certain condition is met (such as "all the functions in the network are working properly"), the connected goals are said to be *achieved*.

In order for a certain function to be available, some goals may have to be achieved. This is expressed by using *condition relations*. These relations connect a goal with a function. There may be several conditions emanating from a goal, and several conditions may be connected to a function.

In order to express management functions, there is also a relation called an *achieve-by-management relation*. This is used to express the fact that the flow functions need some kind of management (automatic or by an operator) to function as designed.

These relations are shown in Figure 3.

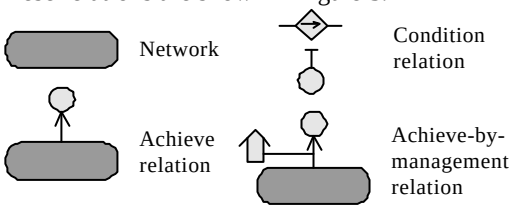

Figure 3. The relations between the MFM objects.

An Example of an MFM Model

To illustrate how MFM is used, an example is presented here. The system, which is shown in Figure 4, consists of an engine, and a cooling system. The cooling fluid is pumped by an electrical pump, and the pump needs lubrication in order to work. The engine runs on gasoline and oxygen, and if the cooling system breaks down, the engine will overheat and stop functioning. The cooling fluid in this system is water.

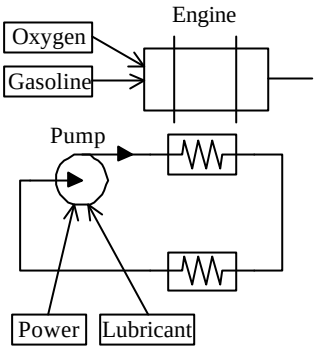

Figure 4. An engine which runs on gasoline and oxygen. The engine needs a cooling system, and the circulator pump in the cooling system needs electrical power and lubricant.

This is not an example taken from a real system, but it is adequate for demonstration purposes. In order to create an MFM model of this system, the important questions to ask are "what are the goals of the system," and "what functions are available in the system." Several goals can be found for this system, but in this case let us concentrate on these five:

- *Keep engine running.* This is the main goal of the system.
- *Supply gas.* Gasoline is needed for the engine to work.
- *Cool engine.* The engine needs cooling in order to work.
- *Lubricate pump.* The pump needs lubrication.
- *Supply power to pump.* The pump needs electrical power.

The functions that can be found in this model are for example "gasoline supply," "gasoline transport," "water transport," and "lubricant source." An MFM model of this system is shown in Figure 5.

The network marked "E1" is a model of the energy balance in the engine. To the left are two sources of energy. The top source is the oxygen supply (the air), and the bottom source is the gasoline supply. In the middle of the network, there is an energy balance where the chemical energy of the gasoline and oxygen is transformed into kinetic energy and heat energy. The kinetic energy is absorbed by the top sink, and the heat energy is absorbed by the bottom sink. If all of these functions work, the top goal ("keep engine running") is achieved.

There are two conditions going up to the top network. The one to the left states that in order for the chemical energy of the gasoline to be available, the gasoline transport from the tank to the engine must be working. The one to the right states that the heat energy sink can not work unless the cooling system is working.

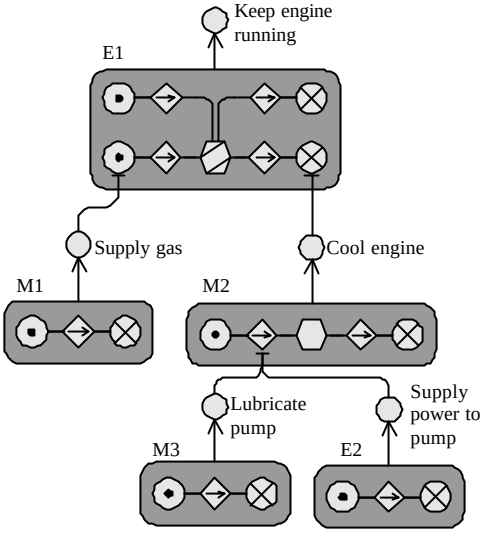

Figure 5. An MFM model of the example system.

The network marked "M1" is a model of the physical transport of gasoline from the tank to the engine. The network marked "M3" is a model of the physical transport of lubricant to the pump. The network marked "E2" is a model of the transport of electrical power to the pump.

The network marked "M2," finally, is a model of the circulation of water in the cooling system. The leftmost transport in the network is a model of the pump, and the storage in the middle is a model of the heat exchanger in the engine. Both the source function and the sink function are models of the other heat exchanger element. This could of course be expressed by modeling the source and the sink in this network as a single storage, and creating a circular flow in the network instead of a straight one, but in this model the straight flow was chosen.

IV. ALGORITHMS FOR MFM

Several algorithms and methods have been developed for use with MFM models. Three of these, *measurement validation*, *fault diagnosis*, and *alarm analysis*, by Larsson [3, 5], have been implemented in the MFM Toolbox, a program for Windows 95 and NT. The MFM Toolbox is a tool for creating, editing and testing MFM models, and is a good base for adding new algorithms.

The measurement validation algorithm uses measured flow values together with the MFM model to check that the flow values are consistent with the model. The fault diagnosis algorithm is similar to a backward chaining expert system, but the inherent structure of an MFM model guarantee that the diagnosis will be completed in linear (or sub-linear) time. This is advantageous for real time systems, see [4]. The algorithms have successfully been used in several projects, see [6, 7].

The alarm analysis algorithm will be described in detail, because the FMA algorithm relies on certain aspects of this algorithm.

Alarm Analysis Using MFM

Most industrial processes are equipped with a large number of sensors. The sensors are all connected to some form of alarm, so that the operators can know when something goes wrong. However, in case of a major fault, many of the alarms may trigger at the same time. This may overload the operators with alarms, and the real cause of the fault may be drowned among all the secondary faults. This is a potentially dangerous situation, and the alarm analysis algorithm for MFM is designed to separate the primary faults from the secondary faults in order to reduce the number of alarms presented to the operator.

The algorithm works by associating discrete alarm states, such as "low flow" and "high volume," with each flow function in the MFM model. Due to the semantic interpretations of the MFM flow functions, it is possible to conclude that certain faults may cause consequential faults in connected flow functions. The method uses a set of causation rules to check each pair of flow functions, to separate the faults that must be primary from the faults that may be consequences of the primary faults. However, the method does not guarantee that the faults marked as secondary are not in fact primary faults. There can be hidden primary faults that look as if they were caused by another primary fault, and which are therefore classified as secondary. Therefore, the algorithm only sorts the faults into the classes "primary" and "secondary or hidden primary."

An example of how the algorithm works is given below. Figure 6 describes a closed tank connected to a pump.

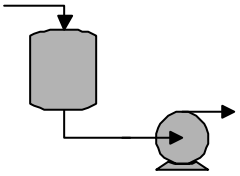

Figure 6. A closed tank, with an inflow and an outflow. The outflow of the tank is connected to a pump.

In this simple system, it is reasonable to assume that a too low volume in the tank may cause a too low flow through the pump. In addition, a too low flow through the pump may cause a too high volume in the tank. There is also the possibility that a too high volume in the tank may cause a too high flow through the pump, and that a too high flow through the pump may cause a too low volume in the tank. These four *causation rules* describe how one fault may cause another.

Now, assume that the tank becomes empty, and therefore a level alarm is activated. Because of the low level in the tank, the pump flow will become too low, and activate another alarm. By using the first rule above, it is possible to conclude that the low volume in the tank is the cause for both alarms, the *primary* fault, whereas the low flow through the pump is a consequential, or *secondary* fault. The alarm analysis algorithm uses a set of rules like these for every legal MFM connection to find out which alarms are primary and which are secondary.

In the MFM representation, the situation described here would be modeled like in Figure 7.

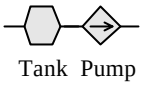

Figure 7. A model of the tank-pump system.

The MFM functions have, as described above, associated alarm states. The storage function can be in one of three alarm states, namely "normal," "high volume" (*hivol*), or "low volume" (*lovol*). The transport function can be in the states "normal," "high flow" (*hiflow*), or "low flow" (*loflow*). The states are set by rules of the following form: "if the flow through the transport is less than a threshold value f_{lo}, set the alarm state of the transport to *loflow*." The threshold value can be statically set in advance, or updated dynamically for example when the state of the process changes.

The causation rules for the tank-pump system can now be written using only MFM functions and their associated alarm states:

1. A storage *lovol* may cause a downstream transport to have a *loflow*.
2. A transport *loflow* may cause an upstream storage to have a *hivol*.
3. A storage *hivol* may cause a downstream transport to have a *hiflow*.
4. A transport *hiflow* may cause an upstream storage to have a *lovol*.

Similar rules have been derived for all legal MFM connections, and they are described in detail in [3, 5]. Since there are only a limited number of legal MFM connections and a discrete number of states, it is possible to store the causation rules in a table. The alarm analysis algorithm looks at each pair of MFM functions, and if both functions are in an alarmed state, the table is used to conclude which function has a primary alarm, and which has a secondary (consequential) alarm. If only one of the two functions is alarmed, the alarmed function is primary.

A special case occurs when a function does not have a connected sensor (the function is said to be *unmeasured*). Since it is then impossible to know the real state of the function, the alarm analysis algorithm uses the state of the connected functions to guess the state of the unmeasured function. This is done using the same table as described above, and the method is called *consequence propagation*. For example, in the situation described above, if the flow through the pump is not measured and the volume in the tank is low, the consequence propagation rules will guess that the flow through the pump is low.

It should be noted that because of the inherent structure of the MFM models, it is always sufficient to only look at

two connected functions at a time. This is important, since it allows the complexity of the algorithm to be kept linear, or even sub-linear, with regard to the size of the model.

V. Failure Mode Analysis using MFM

The failure mode analysis I have developed for MFM is based on the consequence propagation used in the alarm analysis algorithm by Larsson [3], expanded with timing information associated with condition relations and storages. An important fact here is that the consequence propagation in the alarm analysis algorithm will stop as soon as a function with a sensor reading which is in the normal range is found. In the FMA algorithm, the consequence propagation will not stop until the whole model has been searched, because of the assumption that the functions that seem to be working now may stop working in the future. The motives for associating the timing information with only storages and condition relations were:

- Failures will take time to propagate between MFM networks.
- Storages often have an intrinsic capability to delay the propagation of failures within a network.
- Adding time delay information to *all* functions and relations would be unfeasible in terms of modeling effort.

The points above may be debatable, since there are most often propagation delays in other functions as well. However, these objects were carefully chosen because the method will give good results without any serious drawbacks with this setup.

The timing information needed for an analysis must be either derived from simulated cases, derived from stored real process data, or estimated by hand. Often, the last option is the only one available, but the method may result in useful data even if the exact timings are slightly wrong. The same problems arise for all FMA methods that are based on timings, so this is not in any way unique for the MFM version. The method described here has been implemented in a version of the MFM Toolbox, see section IV.

Assignment of Timing Information

When the MFM model is built in the MFM Toolbox, it is possible to assign timing information to each storage and to each condition relation. In the current version of the algorithm, all conditions emanating from the same goal will have the same delay. This is currently under evaluation, and may be changed in the future so that each condition has its own timing delay. The timings are printed next to the function, as shown in Figure 8.

Here, the following is assumed:

- If the gasoline support fails, the engine will stop directly.
- If the engine cooling fails, the engine will run for twenty minutes.
- If the lubrication of the pump stops, the pump will fail after ten minutes.
- If the power to the pump fails, the pump will fail immediately.
- If the pump fails, the stored water in the engines heat exchanger will last for five minutes.

These timings are not taken from any real situation, but they are adequate for this example.

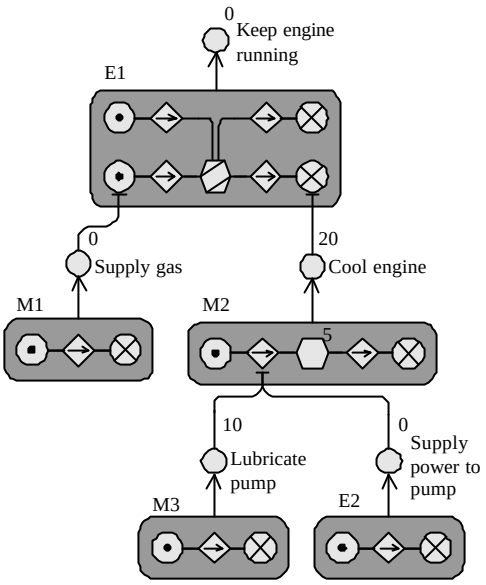

Figure 8. An MFM model of the engine with timing information visible.

An FMA Run

After the timing information is entered into the model, it is possible to run test cases. This is done in the following way: first, the user inputs the start state (the state of the system at time zero), by asserting failures in one or more functions. Then, the analysis is started by a user command, and the result is presented on the screen. An example of this is shown in Figure 9. Here, the initial failure is that the lubricant has run out. The result of this is that the pump will fail due to lack of lubricant, and after a while the engine will fail due to lack of cooling. After the complete FMA run has been performed, the program will collect these failures and generate a list of all functions with alarmed sensors, sorted in order of time-to-failure. This is exemplified in Figure 10.

The Algorithm in Pseudocode

The algorithm works by assigning new state information to each function in the MFM model. The new information consists of a time stamp, called *failure time*, which holds information on when the function will fail. There is also a variable called *current failure time*, which is used during the traversal of the model. It is used to keep track of the current "time" in the system. In pseudocode, the algorithm for failure mode analysis looks like this:

For each function with an alarmed sensor:

- Set the failure time of the function to 0.
- Propagate the alarms as far as possible according to the rules in the alarm analysis algorithm. Set the failure time of the currently examined function to the current failure time, if the function has not been reached before or if the failure time stored for the function is larger than the current failure time.
- Update the current failure time when passing conditions and storages.
- Stop the propagation if the failure time of the currently examined function is less than the current failure time.

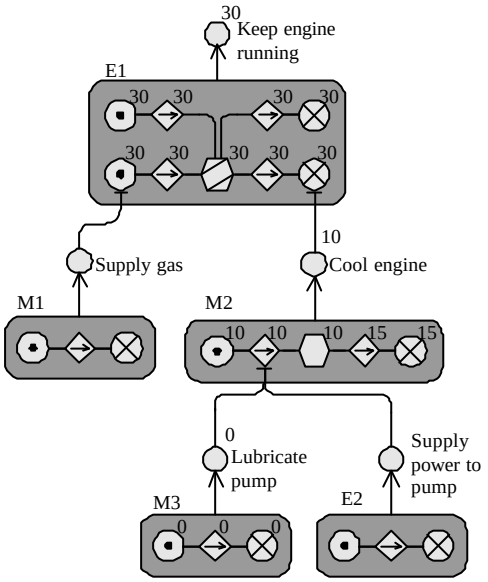

Figure 9. Example of an FMA run. This is a situation where the lubricant runs out, and the impacts of this on the system.

```
Results of Failure Mode Analysis
================================

T = 0
   F8: "Lubricant source"
   F9: "Lubricant transport"

T = 10
   F1: "Water pump"
   F2: "Engine heat exchanger"

T = 30
   F11: "Gasoline supply"
   F17: "Kinetic energy sink"
   F19: "Heat energy sink"
```

Figure 10. The textual output from the FMA run. The measured MFM functions that failed are sorted in time order. The list of functions at time T = 0 are the initially failed functions. Here, the algorithm has predicted that, for example, the water pump will fail within ten minutes.

VI. Conclusions

A new method for failure mode analysis based on MFM has been developed. The method takes a given situation as input, and outputs a list of predicted time-to-failure values for the affected parts of the system. The method has certain advantages compared to traditional methods, for example the possibility to build a system for on-line real time failure prediction with this method. The algorithm has been implemented and tested in the MFM Toolbox, a program for editing and testing MFM models.

VII. Acknowledgments

The author wishes to thank Jan Eric Larsson and Fredrik Dahlstrand at the Department of Information Technology for helpful advice during the development of this algorithm. Thanks also go to Gordon Broderick, Noranda Corporation, for the inspiration that led to the algorithm.

VIII. References

[1] D. J. Allen, M. S. M. Rao, "New Algorithms for the Synthesis and Analysis of Fault Trees," *Industrial and Engineering Chemistry: Fundamentals*, vol. 19, no. 1, pp. 79–85, 1980.

[2] A. Jalashgar, *Identification of Hidden Failures in Process Control Systems through Function-Oriented System Analysis*, Doctor's thesis, Risø–R–936(EN), Risø National Laboratory, Roskilde, Denmark, 1997.

[3] J. E. Larsson, *Knowledge-Based Methods for Control Systems*, Doctor's thesis, TFRT–1040, Department of Automatic Control, Lund Institute of Technology, Lund, 1992.

[4] J. E. Larsson, "Hyperfast Algorithms for Model-Based Diagnosis," Proceedings of the IEEE/IFAC Joint Symposium on Computer-Aided Control Systems Design, Tucson, Arizona, 1994.

[5] J. E. Larsson, "Diagnosis Based on Explicit Means-End Models," *Artificial Intelligence*, vol. 80, no. 1, pp. 29–93, 1996.

[6] J. E. Larsson, B. Hayes-Roth, D. M. Gaba, "Goals and Functions of the Human Body: An MFM Model for Fault Diagnosis," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 27, no. 6, pp. 758–765, 1997.

[7] J. E. Larsson, B. Hayes-Roth, D. M. Gaba, B. E. Smith, "Evaluation of a medical diagnosis system using simulator test scenarios," *Artificial Intelligence in Medicine*, vol. 11, pp. 119–140, 1997.

[8] M. Lind, "Representing Goals and Functions of Complex Systems — An Introduction to Multilevel Flow Modeling," Technical report, 90–D–38, Institute of Automatic Control Systems, Technical University of Denmark, Lyngby, 1990.

[9] M. Modarres, "Functional Modeling of Complex Systems Using a GTST-MPLD Framework," Proceedings of the International Workshop on Functional Modeling of Complex Technical Systems, Ispra, Italy, 1993.

[10] C. J. Price, D. R. Pugh, N. Snooke, J. E. Hunt, M. S. Wilson, "Combining Functional and Structural Reasoning for Safety Analysis of Electrical Designs," *Knowledge Engineering Review*, vol. 12, no. 3, pp. 271–287, 1997.