Failure Mode Modular De-Composition

A mathematical methodology to model and analyse safety critical integrated mechanical/electronic/software systems

Brighton University

PhD Thesis

Version 1.0

Author : R.P. Clark - 2007

Robin Clark
68 Vale Avenue,
Brighton,
East Sussex

Contents

List of Figures

List of Tables

Chapter 1

Thesis Scope

1.1 Introduction

$$\int_0^\infty f(t).e^{-s.t}.dt \mid s \in C$$

This thesis describes the application of, mathematical (formal) techniques to the design of safety critical systems. The initial motivation for this study was to create a system applicable to industrial burner controllers. The methodology developed was designed to cope with both the specific 'simultaneous failures'[3],[**?**],[**?**] and the probability to dangerous fault approach[2].

The visual notation developed was initially designed for electronic fault modelling. However, it could be appleid to mechanical and software domains as well. Due to this a common notation/diagram style can be used to model any integrated safety relevant system.

1.2 Safety Critical Systems

1.2.1 General description of a Safety Critical System

A safety critical system is one in which lives may depend upon it or it has the potential to become dangerous. (/usr/share/texmf-texlive/tex/latex/amsmath/amstext.sty

An industrial burner is typical of plant that is potentially dangerous. An incorrect air/fuel mixture can be explosive. Medical electronics for automatically dispensing drugs or maintaining life support are examples of systems that lives depend upon.

1.2.2 Two approaches : Probablistic, and Compnent fault tolerant

There are two main philosophies applied to safety critical systems. One is a general number of acceptable failure per hour of operation. This is the probablistic approach and is embodied in the european standard EN61508 [2].

The second philosophy, applied to application specific standards, is to investigate components ior sub-systems in the critical safety path and to look at component failure modes and ensure that they cannot cause dangerous faults. With the application specific standards detail specific to the process are This philosophy is first mentioned in aircraft safety operation reseach WWII studies. Here potential single faults (usually mechanical) are traced to catastrophic failures

1.2.3 Overview of regulation of safety Critical systems

reference chapter dealing speciifically with this but given a quick overview.

1.2.3.1 Overview system analysis philosophies

- General safety standards - specific safety standards

1.2.3.2 Overview of current testing and certification

ref chapter speciiffically on this but give an overview now

1.3 Background to the Industrial Burner Safety Analysis Problem

An industrial burner is a good example of a safety critical system. It has the potential for devatating explosions due to boiler overpressure, or ignition of an explosive mixture, and, because of the large amounts of fuel used, is a potential fire hazard. They are often left running unattended 24/7.

To add to these problems Operators are often under pressure to keep them running. An boiler supplying heat to a large greenhouse complex could ruin crops should it go off-line. Similarly a production line relying on heat or steam can be very expensive in production down-time should it fail. This places extra responsibility on the burner controller.

These are common place and account for a very large proportion of the enery usage in the world today (find and ref stats) Industrial burners are common enough to have different specific standards written for the fuel types they usei **??** **??** **??**.

A modern industrial burner has mechanical, electronic and software elements, that are all safety critical. That is to say unhandled failures could create dangerous faults.

To add to these problems Operators are often under pressure to keep them running. An boiler supplying heat to a large greenhouse complex could ruin crops should it go off-line. Similarly a production line relying on heat or steam can be very expensive in production down-time should it fail. This places extra responsibility on the burner controller.

These are common place and account for a very large proportion of the enery usage in the world today (find and ref stats) Industrial burners are common enough to have different specific standards written for the fuel types they usei **??** **??** **??**.

A modern industrial burner has mechanical, electronic and software elements, that are all safety critical. That is to say unhandled failures could create dangerous faults.

A more detailed description of industrial burner controllers is dealt with in chapter **??**.

1.3.1 Mechanical components

describe the mechanical parts - gas valves damper s electronic and software give a diagram of how it all fits A together with a

1.3.2 electronic Components

1.3.3 Software/Firmware Components

1.3.4 A high level Fault Hierarchy for an Industrial Burner

This section shows the component level, leading up higher and higher in the abstraction level to the software levels and finally a top level abstract level. If the system has been designed correctly no ‘undetected faults’ should be present here.

1.4 An Outline of the FMMD Technique

The methodology takes a bottom up approach to the design of an integrated system. Each component is assigned a well defined set of failure modes. The components are formed into modules, or functional groups. These functional groups are analysed with respect to the failure modes of the components. The 'functional group' or module will have a set of derived failure modes. The number of derived failure modes will be less than or equal to the sum of the failure modes of all its components. A 'derived' set of failure modes, is at a higher abstraction level. derived modules may now be used as building blocks, to model the system at ever higher levels of abstraction until the top level is reached.

Any unhandled faults will appear at this top level and will be 'un-resolved'. A formal description of this process is dealt with in Chapter **??**.

Automated systems, as opposed to manual ones are now the norm in the home and in industry.

Automated systems have long been recognised as being more effecient and more accurate than a human opperator, and the reason for automating a process can now be more likely to be cost savings due to better effeciency thatn a human operator **??**.

For instance early automated systems were mechanical, with cams and levers simulating fuel air mixture profile curves over the firing range. Because fuels vary slightly in calorific value, and air density changes with the weather, no optimal tuning can be optional. In fact for asethtic reasons (not wanting smoke to appear at the flue) the tuning was often air rich, causing air to be heated and uneccessarily passed through the burner, leading to direct loss of energy. An automated system analysing the combustions gasses and automatically adjusting the fuel air mix can get the effeciencies very close to theoretical levels.

As the automation takes over more and more functions from the human operator it also takes on more responsibility. A classic example of an automated system failing, is the therac-25. This was an X-ray dosage machine, that, due to software errors caused the deaths of several patients and injured more during the 1980's.

To take an example of an Autopilot, simple early autopilots, were (i.e. they prevented the aircraft staying from a compass bearing and kept it flying striaght and level). Were they to fail the pilot would notice quite quickly and resume manual control of the bearing.

Modern autopilots control all aspects of flight including the engines, and take off and landing phases. The automated system does not have the common sense of a human pilot either, if fed the wrong sensory information it could make horrendous mistakes. This means that simply reading sensors and applying control corrections cannot be enough. Checking for error conditions must also be incorporated. It could also develop an internal fault, and must be able to cope with this.

Systems such as industrial burners have been partially automated for some time. A mechanical cam arrangement controls the flow of air and fuel for the range of firing rate (output of the boiler).

These mechanical systems could suffer failures (such as a mechanical linkage beoming detached) and could then operate in a potentially dangerous state.

More modern burner controllers use a safety critical computer controlling motors to operate the fuel and air mixture and to control the safety valves.

In working in the industrial burner industry and submitting product for North American and European safety approval, it was apparent that formal techniques could be applied to aspects of the ciruit design. Some safety critical circuitry would be subjected to thought experiments, where the actions of one or more components failing would be examined. As a simple example a milli-volt input could become disconnected. A milli-volt input is typically amplified so that its range matches that of the A-¿D converter that you are reading. were this signal source to become disconnected the systems would see a floating, amplified signal. A high impedance safety resistor can be added to the circuit, to pull the signal high (or out of nornal range) upon disconnection. The system then knows that a fault has occurred and will not use that sensor reading (see 1.1).

For exmaple, if the sensor supplies a range of 0 to 40mV, and RG1 and RG2 are such that the op-amp supplies a gain of 100 any signal between 0 and 4 volts on the ADC will be considered in range. Should the sensor become disconnected the opamp will supply its maximum voltage, telling the system the sensor reading is invalid.

Figure 1.1: Milli-Volt Sensor with safety resistor

This introduces a level of self checking into the system. We need to be able to react to not only errors in the process its self, but also validate and look for internal errors in the control system.

This leads on to an important concept of three main states of a safety critical system.

Safety critical systems in the context of this study, means that a safety critical system may be said to be in three distinct overall states. Operating normally, operating in a lockout mode with a detected fault, and operating dangerously with an undetected fault.

The main role of the system designers of safety critical equipment should be to eliminate the possibility of this last condition.

1.5 Motivation for developing a formal methodology

A feature of many safety critical systems specifications, including EN298, EN230 [3] [**?**] is to demand, at the very least that single failures of hardware or software cannot create an unsafe condition in operational plant. Further to this a second fault introduced, must not cause an unsafe state, due to the combation of both faults.

This sounds like an entirely reasonable requirement. But to rigorously check the effect a particular component fault has on the system, we could check its effect on all other components. Should a diode in the powersupply fail in a particular way, by perhaps introducing a ripple voltage, we should have to look at all components in the system to see how they will be affected.

Thus, to ensure complete coverage, each of the effects of the failure modes must be applied to all the other components. Each component must be checked against the failure modes of all other components in the system. Mathematically with components as 'c' and failure modes as 'Fm'.

$$checks = \{\ (Fm, c)\ \mid \hat{c} \neq\ c\} \tag{1.1}$$

Where demands are made for resilience against two simultaneous failures this effectively squares the number of checks to make.

$$doublechecks = \{\ (Fm_1, Fm_2, c)\ \mid\ c_1\ \neq\ c_2\ \wedge\ Fm_1 \neq Fm_2\ \} \tag{1.1}$$

If we consider a system which has a total of N failure modes (see equation 1.1) this would mean

checking a maximum of

$$NumberOfChecks = \frac{N(N-1)}{2} \tag{1.1}$$

for individual component failures and their effects on other components when they fail. For a very small system with say 1000 failure modes this would demand a potential of 500,000 checks for any automated checking process.

European legislation[3] directs that a system must be able to react to two component failures and not go into a dangerous state.

This raises an interesting problem from the point of view of formal modelling. Here we have a binary cross product of all components (see equation 1.1). This increases the number of checks greatly. Given that the binary cross product is $(N^2 - N)/2$ and has to be checked against the remaining $(N - 2)$ components.

$$NumberOfchecks = \frac{(N^2 - N)(N-2)}{2} \tag{1.1}$$

Thus for a 1000 failure mode system, roughly a half billion possible checks would be required for the double simultaneous failure scenario. This astonomical number of potential combinations, has made formal analysis of this type of system, up until now, impractical. Fault simulators are commonly used for the gas certification process. Thus to manually check this number of combinations of faults is in practise impossible. A technique of modularising, or breaking down the problem is clearly necessary.

1.6 Challenger Disaster

One question that anyone developing a safety critical analysis design tool could do well to answer, is how the methodology would cope with known previous disasters. The Challenger disaster is a good example, and was well documented and invistigated.

The problem lay in a seal that had an operating temperature range. On the day of the launch the temperature of this seal was out of range. A bottom up safety approach would have revealed this as a fault.

1.7 Problems with Natural Language

Written natural language desciptions can not only be ambiguous or easy to misinterpret, it is also not possible to apply mathematical checking to them.

A mathematical model on the other hand can be checked for obvious faults, such as tautologies and contradictions, but also intermediate results can be extracted and these checked.

Mathematical modeling of systems is not new, the Z language has been used to model systems[**?**]. However this is not widely understood or studied even in engineering and scientific circles. Graphical techniques for representing the mathematics for specifying systems, developed at Brighton and Kent university have been used and extended by this author to create a methodology for modelling complex safety critical systems, using diagrams.

This project uses a modified form of euler diagram used to represent propositional logic.

1.8 Ideal System Designers world

Imagaine a world where, when ordering a component, or even a complex module like a a failsafe sensor/scientific instrunment, one page of the datasheet is the failure modes of the system. All possible ways in which the component can fail and how it will react when it does.

1.8.1 Environmentally determined failures

Some systems and components are guaranteed to work within certain environmental constraints, temperature being the most typical. Very often what happens to the system outside that range is not defined. Where this is the case, these are undetectable errors.

1.9 Project Goals

- To create a user friendly formal common visual notation to represent fault modes in Software, Electronic and Mechanical sub-systems.
- To formally define this visual language.
- To prove that tehe modules may be combined into hierarchies that truly represent the fault handling from component level to the highest abstract system 'top level'.
- To reduce to complexity of fault mode checking, by modularising and building complexity reducing hierarchies.
- To formally define the hierarchies and procedure for bulding them.
- To produce a software tool to aid in the drawing of diagrams and ensuring that all fault modes are addressed.
- To allow the possiblility of MTTF calculation for statistical reliability/safety calculations.

Chapter 2

An overview of European and North Americans Standards

Chapter 3

Statistical Methods and Models

Chapter 4

Survey of Safety Critical Analysis Metyhodologies and Tools Available

Chapter 5

Propositional Logic Diagrams

Abstract

Propositial Logic Diagrams have been designed to provide an intuitive method for visualising and manipulating logic equations, to express fault modes in Mechanical and Electronic Systems. Diagrams of this type can also be used to model the logical conditions that control the flow of a computer program. This type of diagram can therefore integrate logical models from mechanical, electronic and software domains. Nearly all modern safety critical systems involve these three disiplines. It is intended to be used for analysis of automated safety critical systen Many types of safety critical systems now legally require fault mode effects analysis[**?**], but few formal systems exist and wide-spread take-up is not yet the norm.[**?**]. Because of its visual nature, it is easy to manipulate and model complicated conditions that can lead to dangerous failures in automated systems.

The Diagrams described here form the mathematical basis for a new visual and formal system for the analysis of safety critical software and hardware systems.

5.1 Introduction

Propositional Logic Diagrams (PLDs) have been devised to collect and simplfy fault modes in safety critical systems undergoing static analysis[**?**][**?**].

This type of analysis treats failure modes within a system as logical states. PLD provides a visual method for modelling failure mode analysis within these systems, and specifically identifying common failure symptoms in a user friendly way. Contrasting this to looking at many propositional logic equations directly in a text editor or spreadsheet, a visual method is percieved as being more intuitive.

PLDs use three visual features that can be combined to represent logic equations. Closed contours, test cases, and lines that link test cases. All features may be labelled, and the labels must be unique within a diagram, however contours may be repeated in the diagram.

Test cases are marked by asterisks. These are used as a visual 'anchor' to mark a logical condition, the logical condition being defined by the contours that enclose the region on which the test case has been placed. The contours that enclose represent conjuction. Test cases may be connected by joining lines. These lines represent disjunction (Boolean 'OR') of test cases.

With these three visual syntax elements, we have the basic building blocks for all logic equations possible.

Test cases - Points on the plane indicating a logical condition.

Conjunction - Overlapping contours

Disjunction - Joining of named test cases.

5.2 Formal Description of PLD

Definitions of conrete and abstract PLD's follow. Well-formedness conditions for PLD's are separated from this definition, because of practical differences between the way they are used to represent software as opposed to representing electronics and mechanical systems.

5.2.1 Concrete PLD Definition

A concrete *Propositional logic diagram* is a set of labeled *contours* (closed curves) in the plane. The minimal regions formed by the closed curves can by occupied by 'test points'. The 'test points' may be joined by joining lines. A group of 'test points' connected by joining lines is defined as a 'test point disjunction' or Spider. Spiders may be labeled.

To differentiate these from common Euler diagram notation (normally used to represent set theory) the curves are drawn using dotted and dashed lines.

5.2.2 PLD Definition

In English: The elements that can be found in a PLD diagram are a number of contours, a number of test points and joining lines that connect test points.

Definition: 1. *A concrete PLD d is a set comprising of a set of closed curves $C = C(d)$, a set of test points $T = T(d)$ and a set of test point joining lines $J = J(d)$.*

$$d = \{C, T, J\}$$

In English: Each element of the diagram has a unique label within the diagram.

Definition: 2. *A minimal region of concrete PLD diagram d is a connected component of*

$$\mathbb{R}^2 - \bigcup_{\hat{c} \in \hat{C}(\hat{d})} \hat{c}$$

Definition: 3. *Let d be a PLD and $\mathcal{X} \subseteq \hat{C}(\hat{d})$ a set of countours. If the set*

$$\hat{z} = \bigcap_{c \in \mathcal{X}} interior(\hat{c}) \cup \bigcap_{\hat{c} \in \hat{C} - X} exterior(\hat{c})$$

is non empty, then $\hat{z}$ is a concrete zone of $\hat{d}$. A zone is a union of minimal regions. The set of all concrete zones of $\hat{d}$ is denoted $\hat{\mathcal{Z}}$.

Each minimal region in the plane may be inhabited by one or more 'test points'. Each test point can be associated with the set of contours that enclose it.

Definition: 4. *$\mathcal{Z}_d : T(d) \rightarrow \mathcal{C}$ is a function associating a testpoint with a set of contours in the plane. This corresponds to the interior of the contours defining the zone.*

Pairs of test points may be joined by joining lines. The operator $\overset{join}{\leftrightarrow}$ is used to show that two points are joined by a line in the concrete diagram.

Definition: 5. *$\mathcal{F}_j$ is a function associating a joining line with a pair of test points. The Join t1,t2 is defined as*

$$\mathcal{F}_d : J(d) \rightarrow \{t1, t2 \mid t1 \in T(d) \wedge t2 \in T(d) \wedge t1 \neq t2\}$$

In English: Test points on the concrete diagram pair-wise connected by a 'joining line'

A collection of test points connected by joining lines, is an Fuctionally Merged Group, FMG or 'test point disjunction'. An FMG has members which are test points.

may be merged and create a

Definition: 6. *Let d be a PLD : An FMG is a maximal set of test points in d where the test points belong to a sequence connected by joining lines such that:*

$$t_i \overset{join}{\leftrightarrow} t_n, for\ i = 1, ..., n$$

OR consider an FMG as a tree whose nodes are test points.

A singleton test point can be considered a sequence of one test point and is therefore also an FMG.

5.2.3 Semantics of PLD

- A closed curve in a PLD represents a condition (logical state) being modelled.
- A test point represents the conjunction of the conditions represented by the curves that enclose it.
- A FMG represents the disjunction of all test points that are members of it.

To obtain the set of propositions from a PLD, each FMG must be processed. For each test case in the FMG a new section of the equation is disjuctively appended to it.

Let conjunctive logic equation associated with a test point be determined from the contours that enclose it. i.e. the contours $\mathcal{X}$ from the zone it inhabits.

Definition: 7. *Let $\mathcal{F}_t$ be a function mapping a test point to a proposition / logical equation $p \in P$. The test point inhabits the zone $\mathcal{Z}$ which is a collection of contours (the contours that enclose the test point.*

$$\mathcal{F} : T \rightarrow P$$

$$\mathcal{F}(t) : p = \bigwedge_{c \in \mathcal{Z}} c$$

In English: Thus a 'test point' enclosed by contours labelled a, b, c would be represented by the logic equation $a \wedge b \wedge c$.

Definition: 8. *Let $\mathcal{G}_{fmg}$ be a function that returns a logic equation for a given FMG fmg in the diagram, where an FMG is a non empty set of test points*

$$\mathcal{G} : FMG \rightarrow P_{fmg}$$

The logic equation representing an FMG p_{fmg} can be determined thus.

$$\mathcal{G}_{fmg}(fmg) = \bigvee_{t \in fmg} (\mathcal{F}_t(t))$$

The abstract PLD diagram is a set of logic equations representing all FMGs, along with unused zones (i.e. zones that are not inhabited by FMGs).

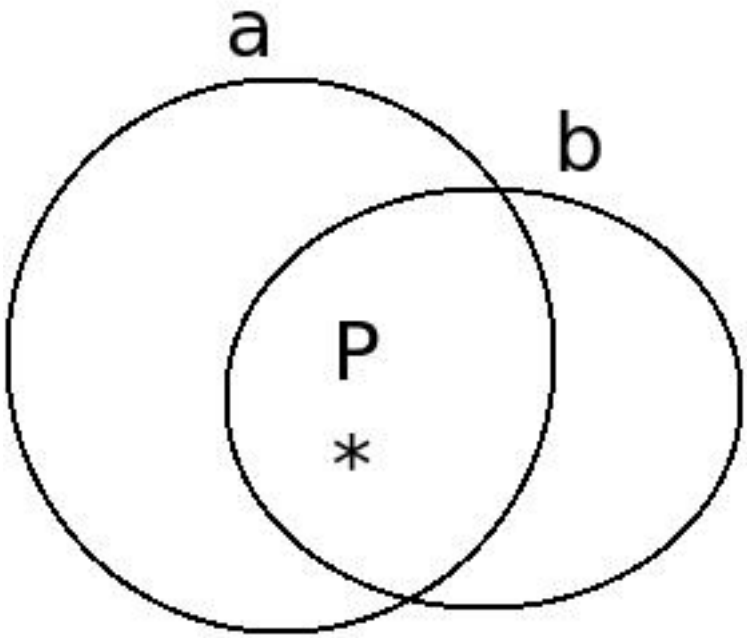
Definition: 9. *A diagram can be reduced to a collection of FMGs. A new diagram can be derived from this, replacing a contour for each FMG. This diagram is at one higher level of abstraction then the diagram that it was produced from.*

5.3 Example Diagrams

5.3.1 How to read a PLD diagram

PLD diagrams are read by first looking at the test case points. The test case asterisk will be enclosed by one or more contours. These contours are collected and form the logical conjunction equation for the test case. These test case points thus represent the conjunctive aspects of an equation defined in a PLD. Where these test cases are joined by lines; these represent disjunction of the conjunctive aspects defined by the test cases. Joining lines thus represent dis-junction in a PLD.

5.3.2 Logical AND example



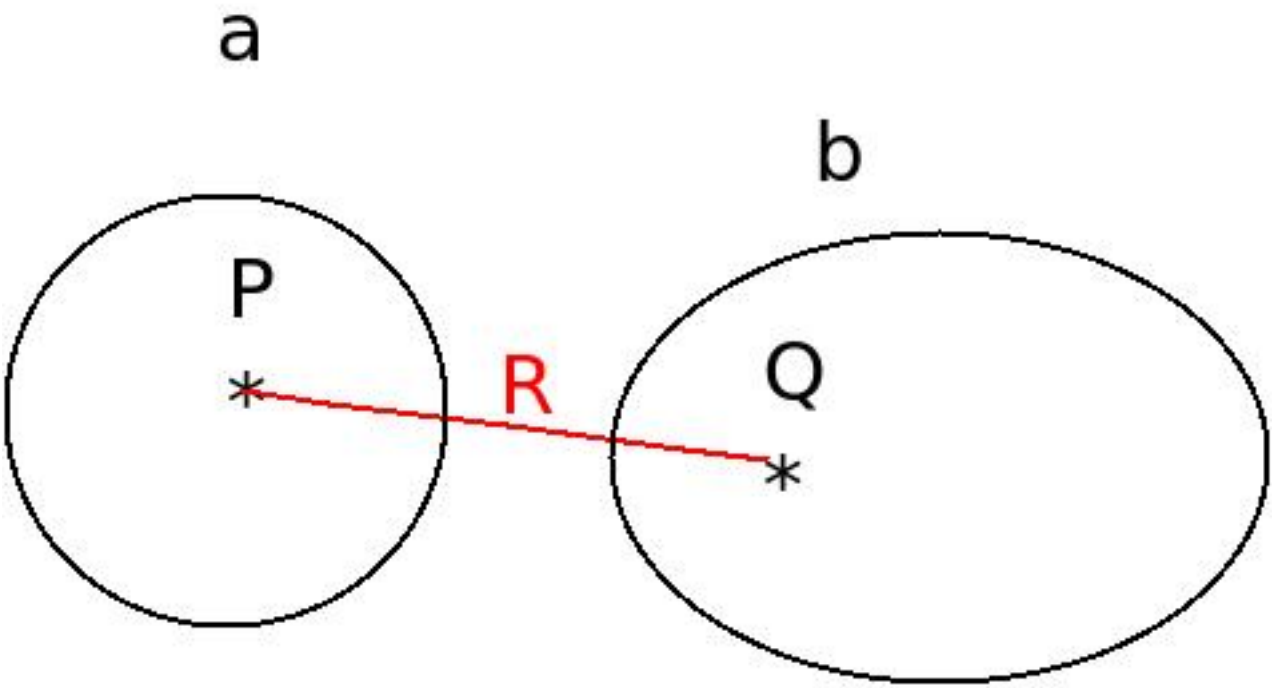
In the diagram 5.3.2 the area of intersection between the contours a and b represents the conjunction of those conditions. The point P represents the logic equation

$$P = (a \wedge b)$$

There are no disjunctive joining lines and so this diagram represents one equation only, $P = (a \wedge b)$.

How this would be interpreted in failure analysis In failure analysis, this could be considered to be a sub-system with two failure states a and b. The proposition P considers the scenario where both failure modes are active.

5.3.3 Logical OR example



The diagram 5.3.3 is converted to Boolean logic by first looking at the test cases, and the contours they are placed on.

$$P = (a)$$

$$Q = (b)$$

The two test cases are joined by a the line named R. we thus apply disjunction to the test cases.

$$R = P \vee Q$$

substituting the test cases for their Boolean logic equations gives

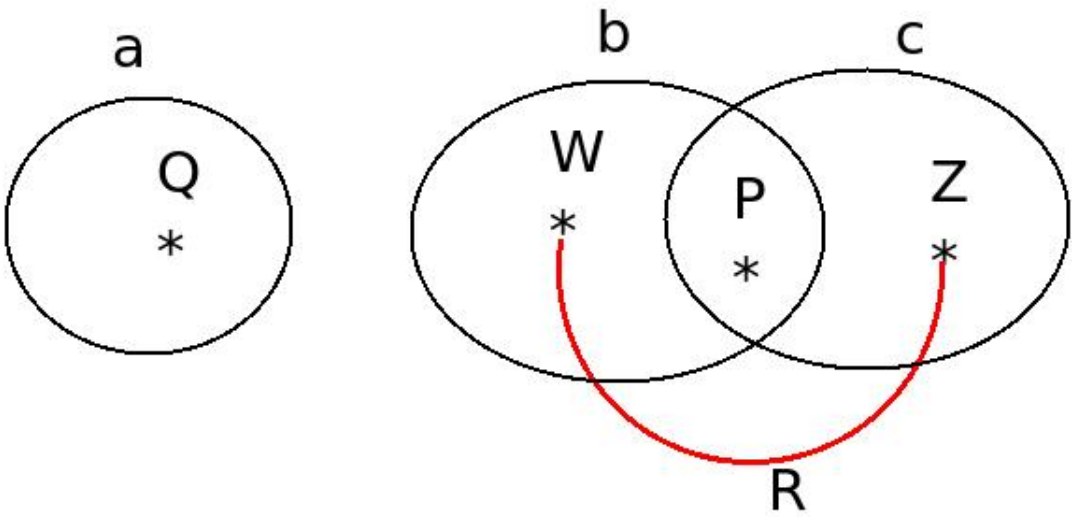
$$R = ((a) \vee (b))$$

.

5.3.4 Labels and useage

In diagram **??** Z and W were labeled but were not necessary for the final expression of $R = b \vee c$. The intended use of these diagrams, is that resultant logical conditions be used in a later stage of reasoning. Test cases joined by disjunction, all become represented in one, resultant equation. Therefore only test cases not linked by any disjunctive joining lines need be named.

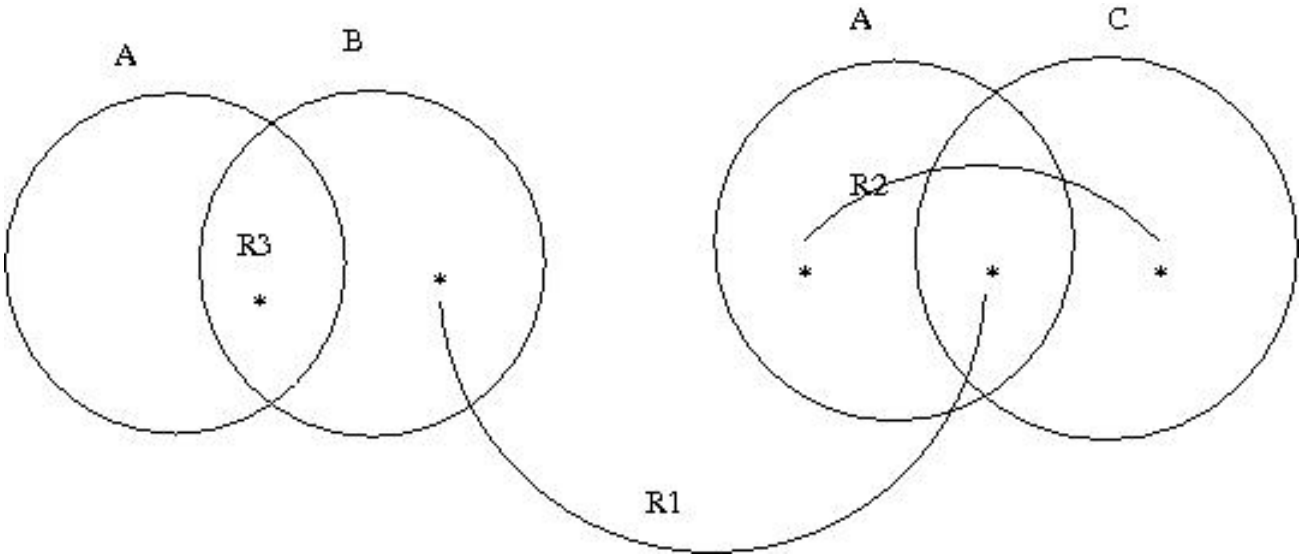
The diagram **??** can therefore be represented as in diagram 5.3.4, with two unnamed test cases.



How this would be interpreted in failure analysis In failure analysis, this could be considered to be a sub-system with two failure states a and b. The proposition P considers the scenario where either failure mode is active. Additionally it says that either failure mode a or b being active will have a resultant effect R on the sub-system. Note that the effect of a and b both being active is not defined on this diagram.

5.3.5 Repeated Contour example

Repeated contours are allowed in PLD diagrams. Logical contradictions or tautologies can be detected automatically by a software tool which assists in drawing these diagrams.



The diagram 5.3.5 is converted to Boolean logic by first looking at the test cases, and the contours they are placed on.

$$P = (b)$$

$$Q = (a) \wedge (c)$$

The two test cases are joined by a the line named $R1$. we thus apply disjunction to the test cases.

$$R1 = P \vee Q$$

$$R1 = b \vee (a \wedge c)$$

.

$R2$ joins two other test cases

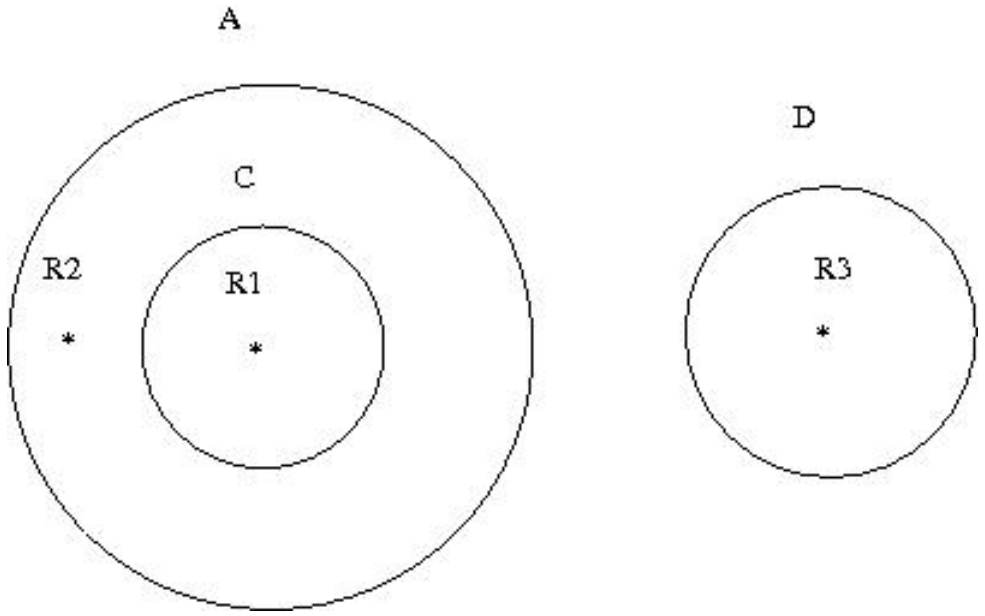
$$R2 = a \vee c$$

The test case residing in the intersection of countours B and A represents the logic equation $R3 = a \wedge b$.

How this would be interpreted in failure analysis In failure analysis, $R2$ is the symptom of either failure mode A or C occurring. $R1$ is the symptom of B or $A \wedge C$ occurring. There is an additional symptom, that of the test case in $A \wedge B$.

5.3.6 Inhibit Failure

Very often a failure mode can only occurr given a searate environmental condition. In Fault Tree Analysis (FTA) this is represented by an inhibit gate.



The diagram 5.3.6 has a test case in the contour C. Contour C is enclosed by contour A. This says that for failure mode C to occur failure mode A must have occurred. A well known example of this is the space shuttle 'O' ring failure that caused the 1986 challenger disaster [1]. For the failure mode to occurr the ambiant temperature had to be below a critical value. If we take the failure mode of the 'O' ring to be C and the temperature below critical to be A, we can see that the low temperature failure mode C can only occurr if A is true. The 'O' ring could fail in a different way independant of the critical temperature and this is represented, for the sake of this example, by contour D.

In terms of propositional logic, the inhibit gate of FTA, and the countour enclosure of PLD represent *implication.*

c	a	$R1$
F	F	T
F	T	T
T	F	F
T	T	T

$$R1 = c \implies a$$

$$R2 = a$$

$$R3 = d$$

How this would be interpreted in failure analysis In failure analysis, $R2$ is the symptom of either failure mode A or C occurring. $R1$ is the symptom of B or $A \wedge C$ occurring. Note that although R2 is a symptom of the sub-system, on its own it will not lead to a dangerous failure mode of the subsystem.

5.4 Intended use in FMMD

The intention for these diagrams is that they are used to collect component faults and combinations thereof, into faults that, at the module level have the same symptoms.

5.4.1 Example Sub-system

For instance were a 'power supply' being analysed there could be several individual component faults or combinations that lead to a situation where there is no power. This can be described as a state of the powersupply being modeelled as NO_POWER. These can all be collected by DISJUCNTION, i.e. that this this or this fault occuring will cause the NO_POWER fault. Visually this disjuction is indicated by the joining lines. As far as the user of the 'power supply' is concerned, the power supply has failed with the failure mode NO_POWER. The 'power supply' module, after this process will have a defined set of fault modes and may be considered as a component at a higher level of abstraction. This module can then be combined with others at the same abstraction level. Note that because this is a fault collection process the number of component faults for a module must be less than or equal to the sum of the number of component faults.

`CVS Revision Identity $Id: logic_diagram.tex,v 1.17 2010/01/06 13:41:32 robin Exp $`

Compiled last January 15, 2010

Chapter 6

Electronic Components as PLDs

Chapter 7

Software as PLDs

Chapter 8

Mechanical Sub-systems as PLDs

Chapter 9

Symptom Extraction

Abstract

In modular systems design, it is often very useful to know the failure modes of the sub-systems used. This paper outlines a technique for determining the failure modes of a sub-system given its component parts.

The technique uses a graphical notation, based on Euler[**?**] and Constraint diagrams[**?**] to model failure modes and failure mode common symptom collection. The technique is designed for making building blocks for a hierarchical fault model.

Once the failure modes have been determined for a sub-system, that sub-system may be treated as a 'component' or 'black box' and used in conjunction with other such analysed sub-systems, to model higher level sub-systems. In this way a hierarchy to represent the fault behaviour of a system can be built.

The hierarchy is built from the bottom up. Starting with component failure modes at the bottom. Because the process is bottom-up no component failure mode can be overlooked. Once a hierarchy is in place it can be converted into a fault data model.

From the fault data model, automatic generation of FTA[**?**] (Fault Tree Analysis) and mimimal cuts sets[**?**] are possible. Also statistical reliability[**?**] and MTTF (Mean Time to Failure) calculations can be produced automatically, where component failure mode statistics are available[**?**].

This paper focuses on the process of building the blocks that are used in the hierarchy.

9.1 Introduction

9.1.1 Static Analysis

In the field of safety critical engineering; to comply with European Law a product must be certified under the approriate 'EN' standard. Typically environmental stress, EMC, electrical stressing, endurance tests, software inspections and project management quality reviews are applied[**?**].

Static testing is also applied. This is theoretical analysis of the design of the product from the safety perspective. Three main techniques are currenly used, Statistical failure models, FMEA (Failure mode Effects Analysis) and FTA (Fault Tree Analysis). The technique outlined here aims to provide a mathematical frame work to assist in the production of these three results of static analysis.

The aims are

- To automate the process where possible
- To apply a documented trail for each analysis phase (determination of functional groups, and analysis of component failure modes on those groups)
- To use a modular approach so that analysed sub-systems can be re-used
- Automatically ensure no failure mode is unhandled

- To produce a data model from which FTA, FMEA and statistical failure models may be obtained automatically

9.1.2 Systems, functional groups, sub-systems and failure modes

It is helpful here to define some terms, 'system', 'functional group', 'component', 'base component' and 'sub-system'.

A System, is really any coherent entity that would be sold as a safety critical product. A sub-system is a system that is part of some larger system. For instance a stereo amplifier separate is a sub-system. The whole Sound System, consists perhaps of the following 'sub-systems': CD-player, tuner, amplifier separate, loudspeakers and ipod interface.

Thinking like this is a top down analysis approach and is the way in which FTA[**?**] analyses a System and breaks it down.

A sub-system will be composed of component parts, which may themselves be sub-systems. However each 'component part' will have a fault/failure behaviour and it should always be possible to obtain a set of failure modes for each 'component'.

If we look at the sound system again as an example; the CD player could fail in serveral distinct ways, no matter what has happened to it or has gone wrong inside it.

Using the reasoning that working from the bottom up forces the consideration of all possible component failures (which can be missed in a top down approach) we are presented with a problem. Which initial collections of base components should we choose ?

For instance in the CD player example; to start at the bottom; we are presented with a massive list of base components, resistors, motors, user switches, laser diodes all sorts ! Clearly, working from the bottom up we need to pick small collections of components that work together in some way. These are termed 'functional groups'. For instance the circuitry that powers the laser diode to illuminate the CD might contain a handful of components, and as such would make a good candidate to be one of the base level functional groups.

In choosing the lowest level (base component) sub-systems we would look for the smallest 'functional groups' of components within a system. A functional group is a set of components that interact to perform a specific function.

When we have analysed the fault behaviour of a functional group, we can treat it as a 'black box'. We can now call our functional group a sub-system. We know how will behave under fault conditions ! This type of thinking is starting to become more commonplace in product literature, with the emergence of reliability safety standards such as IOC1508[**?**],EN61508[**?**]. FIT (Failure in Time - expected number of failures per billion hours of operation) values are published for some micro-controllers. A micro controller is a complex sub-system in its self and could be considered a 'black box' with a given reliability. [1]

As electrical components have detailed datasheets a useful extension of this would be failure modes of the component, with environmental factors and MTTF statistics.

Currently this sort of information is generally only available for generic component types[**?**].

Notes:

[1]Microchip sources give an FIT of 4 for their PIC18 series micro controllers[**?**], The DOD 1991 reliability manual[**?**] applies a FIT of 100 for this generic type of component

Definition	*Description*
System	A product designed to work as a coherent entity
Sub-system	A part of a system, sub-systems may contain sub-systems
Functional Group	A collection of sub-systems and/or components that interact to perform a specific function
Base Component	Any bought in component, which hopefully has a known set of failure modes
Failure mode	A way in which a System, Sub-system or component can fail

9.2 The Symptom abstraction Process

symptom abstraction described The objective of 'symptom abstraction' is to analyse the functional group and find out what will happen to it, when specified component failure modes occur. Once we know how it fails as a functional group, we can treat it as a component or sub-system with its own set of failure modes.

Each failure mode (or combination of) investigated is termed a 'test case'. Each 'test case' is analysed. The component failure modes are examined with respect to their effect on the functional group. When all 'test cases' have been analysed a second phase is applied.

This looks at the results of the 'test cases' as symptoms of the sub-system. In this way 'test case results' are grouped as common symptoms, from the perspective of the sub-system. To go back to the CD player example, a failed output stage, and a failed internal audio amplifier, will both cause the same failure; *no_sound* !

symptom abstraction represented on the diagram This process can be applied using a diagram. From the collection of parts for the sub-system under analysis, a set of failure modes for each component is obtained. A diagram is then drawn with each component failure mode represented by a contour. Component failure mode combinations are chosen for 'test cases'.[2]

A 'test case' is represented on the diagram as a point or asterisk, in a region enclosed by the contours representing the failure modes it investigates.

The effect on the sub-system of each test case is analysed. The 'test case results' are archived. When all test cases have been analysed, we switch our attention to a higher abstraction level.

We can now try to simplfy by determining common symptoms. A common symptom, in this context, is defined as faults caused by different component failure modes that have the same effect from the perspective of a 'user' of the sub-system.

Test case results can now viewed as failure modes of the sub-sytem or 'black box', and grouped together where there are common symptoms. These are grouped together by joining them with lines. These lines form collected groups (or 'spiders'). See figure 9.4. It can be seen now that each *lone test case* and *spider* on the diagram is a distinct failure mode of the sub-system. This means that these failure modes represent the fault behaviour of the sub-system. We can now treat this sub-system as a component in its own right, or in other words, we have derived a failure mode model at a higher level of abstraction.

We can now draw a new diagram to represent the failure modes of the sub-system. Each spider or lone test case, becomes a contour representing a failure mode of the sub-system in this new diagram (see figure 9.4.

Notes:

[2]Combinations of component failure modes can be represented by overlapping contours

9.3 The Process : To analyse a base level sub-system

To sumarise:

- Determine a minimal functional group
- Obtain list of components in the functional group
- Collect the failure modes for each component
- Draw these as contours on a diagram
- Where multiple failures are examined use overlapping contours
- For each region on the diagram, make a test case
- Examine each test case and determine the effect of the component failure modes on the behaviour of the functional group
- Collect common symptoms. Imagine you are handed this functional group as a 'black box', a sub-system to use. Determine which test cases produce the same fault symptoms. Join common symptoms with lines connecting them (sometimes termed a 'spider').
- The lone test cases and the spiders are now the fault mode behaviour of the sub-system.
- A new diagram can now be drawn where each spider, or lone test case from the original diagram is represented as a contour. These contours represent the failure modes of the sub-system.

9.4 A general Sub-System example

Consider a functional group FG with component parts A,B and C. Each part has a set of related fault modes (i.e. ways in which it can fail to operate correctly). Let us define the following failure modes for each component part, defining a function $FM()$ where K is a component part and F is its set of failure modes[3].

$$FM : K \mapsto F$$

For our example above

$$FM(A) = \{a_1, a_2, a_3\}$$
$$FM(B) = \{b_1, b_2\}$$
$$FM(C) = \{c_1, c_2\}$$

We can now represent the sub-system as a set of component faulure modes FG_{cfm}, thus

$$FG_{cfm} = \{a_1, a_2, a_3, b_1, b_2, c_1, c_2\} \tag{9.0}$$

The failure modes of the components can be represented as contours on on the diagram in 9.4.

We can now look at the effects that component failure modes have on the sub-system. This process involves examining 'test cases'. Each 'test case' represents the fault behaviour of the sub-system due to particular combinations of component fault modes.

Each test case can be represented on the diagram as a labeled point. The labeled point will reside in a region on the diagram enclosed by the contours representing particular component fault modes. The

Notes:

[3]Base component failure modes are defined, often with statistics and evironmental factors in a variety of sources. [**?**]

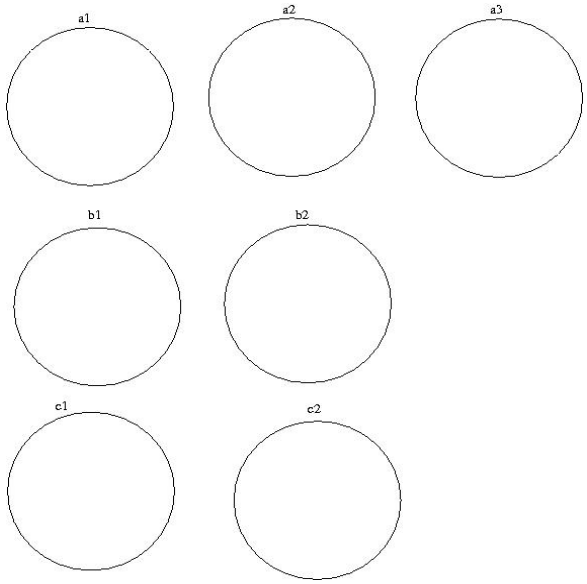


Figure 9.1: FG_{cfm} Component Failure modes represented as contours

label will indicate the fault symptom from the perspective of the sub-system. For the sake of example, only single component failure modes are considered. We can now assign a test case to each contour, and mark it on the diagram.

Component Failure Mode	*test case*
a_1	fs_1
a_2	fs_2
a_3	fs_3
b_1	fs_4
b_2	fs_5
c_1	fs_6
c_2	fs_7

The sub-system fault symptoms are now represented on the diagram as in figure 9.4.

A second stage of analysis is now applied. Empirically, it is often noticed that a sub-system will fail in the same way due to a variety of reasons. To the 'user' of the sub-system, it does not matter which component or combination of components has failed. The sub-system can thus be considered to have its own set of failure modes. This stage of the analysis is to determine these, to collect 'like symptoms'. This is performed on the diagram by linking the test cases with lines to form 'spiders'

For the sake of example let us consider the fault symptoms $SP1 = \{fs_2, fs_4, fs_5\}$ to be an identical failure mode at the *sub-system* level. These can then be joined to form a spider. Likewise let $SP2 = \{fs_1, fs_3, fs_7\}$ be an identical failure mode at the *sub-system* level. Let $\{fs_6\}$ be a distinct failure mode at *sub-system* level.

The diagram can now be drawn as in figure 9.4.

The third stage of the process can be applied automatically. Each 'spider' or 'lone test case' becomes a contour in the new diagram (see figure 9.4.

The result of this will be, a set of failure modes for the sub-system, as though it were a *black box* or a *component* to be used in higher level designs.

We have now in $SP1$, $SP2$ and fs_6 the three ways in which this sub-system can fail. In other words we have derived failure modes for this sub-system.

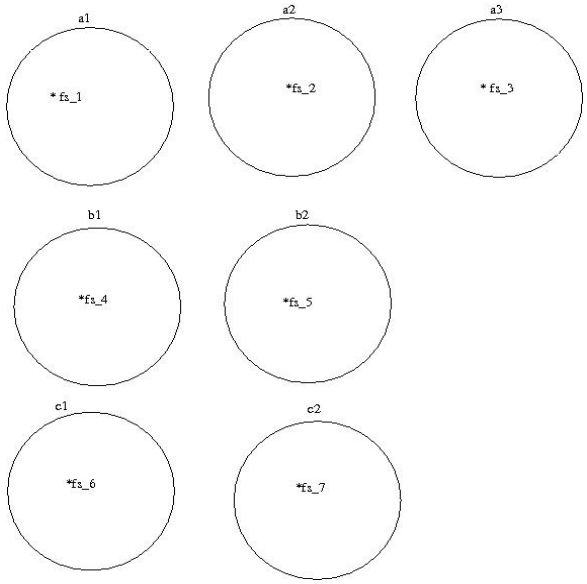


Figure 9.2: Component Failure modes with analysed test cases

This sub-system may now therfore, be represented as three separate failure modes. We may now treat this sub-system as we would a component with a known set of failure modes. The failure modes of the Sub-system SS are now the set $SS_{fm} = \{SP1, Sp2, fs_6\}$.

Defining the function '$\bowtie$' to represent the *symptom abstraction* process, we may now write

$$\bowtie : SubSystemComponentFaultModes \mapsto SubSystemFaultModes$$

$$\bowtie (FG_{cfm}) = SS_{fm} \tag{9.0}$$

The SS_{fm} set of fault modes can be represented as a diagram with each fault mode of SS being a contour. The derivation of SS_{fm} is represented graphically using the '$\bowtie$' symbol, as in figure 9.4

The derived diagram in figure 9.4 shows the functional group of components A, B, C analysed as a sub-system. The result is a set of fault modes that define the fault mode behaviour of that sub-system.

This sub-system, with its three error modes, can now be treated as a component (although at a higher level of abstraction) with known failure modes.

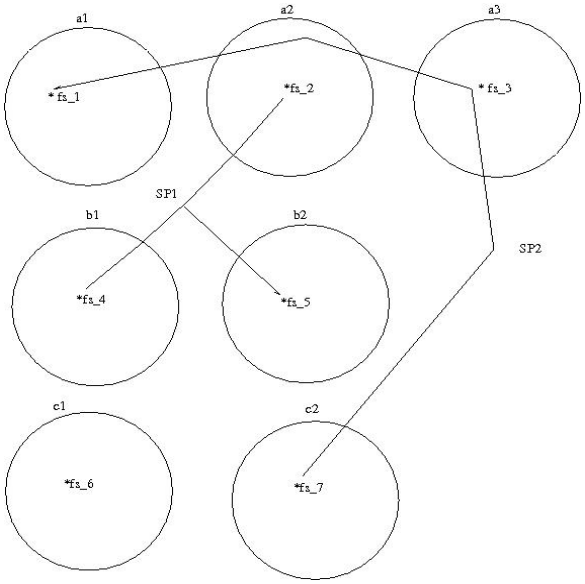


Figure 9.3: Common failure modes collected as 'Spiders'

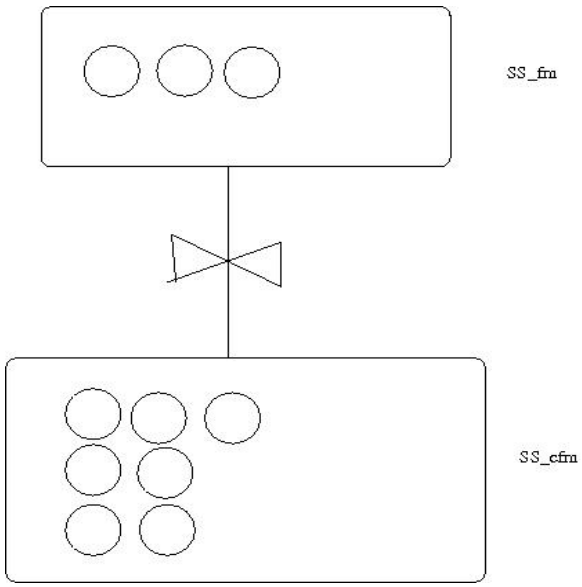


Figure 9.4: Deriving a new diagram

9.5 A Formal Algorithmic Description of 'Symptom Abstraction'

The algorithm for *symptom abstraction* is described in this section using set theory.

The *symptom abstraction process* (given the symbol '$\bowtie$') takes a functional group FG and converts it to a sub-system SS. The sub-system SS is a collection of failure modes of the sub-system. The sub-system SS may now be treated as a component with a known set of failure modes. Thus SS can be used as a system building block at a higher level of fault abstraction.

The algorithm has been broken down into five stages, each following on from the other.

Algorithm 1 Determine failure modes: $FG \mapsto FG_{cfm}$

1: Let FG be a set of components { The functional group should be chosen to be minimally sized collections of components that perform a specific function}
2: Let c represent a component
3: Let CFM represent a set of failure modes
4: $FM(c) \mapsto CFM$ {Let the function FM take a component and return a set of all its failure modes}

Ensure: Each component $c \in FG$ has a known set of failure modes i.e. $FM(c) \neq \emptyset$

5: let FG_{cfm} be a set of failure modes
6: Collect all failure modes from the components into the set FM_{cfm}

Algorthim 1 has taken a functional group FG and returned a set of failure modes FG_{cfm}. The next task is to formulate 'test cases'. These are the collections of failure modes that will be used in the analysis stages.

Algorithm 2 Determine Test Cases: $FM_{cfm} \mapsto TC$

Require: Determine the test cases to be applied

1: All test cases are now chosen by the investigating engineer(s). Typically all single component failures are investigated with some specially selected combination faults
2: Let TC be a set of test cases
3: Let tc_j be set of component failure modes where j is an index of J { Each set tc_j is a 'test case' }
4: $\forall j \in J | tc_j \in TC$
{ Ensure the test cases are complete and unique }
5: **for all** $tc_j \in TC$ **do**

Ensure: $tc_j \in \mathcal{P}FG_{cfm}$ { require that the test case is a member of the powerset of FM_{cfm} }

Ensure: $\forall\, j2 \in J(\forall\, j1 \in J | tc_{j1} \neq tc_{j2} \wedge j1 \neq j2)$ { Test cases must be unique }

6: **end for**
7: let f represet a component failure mode

Require: That all failure modes are represented in at least one test case

Ensure: $\forall f | (f \in FM_{cfm}) \wedge (f \in \bigcup TC)$ { This corresponds to checking that at least each failure mode is considered at least once in the analysis; some european standards imply checking all double fault combinations[**?**] }

Algorithm 2 has taken the set of failure modes FM_{cfm} and returned a set of test cases TC. The next stages is to analyse the effect of each test case on the functional group.

Algorithm 3 Analyse Test Cases: $TC \mapsto R$

1: let r be a 'test case result'
2: Let the function $Analyse : tc \mapsto r$ { This analysis is a human activity, examining the failure modes in the test case and determining how the functional group will fail under those conditions}
3: R is a set of test case results $r_j \in R$ where the index j corresponds to $tc_j \in TC$
4: **for all** $tc_j \in TC$ **do**
5: $rc_j = Analyse(tc_j)$ {this is Fault Mode Effects Analysis (FMEA) applied in the context of the functional group}
6: $rc_j \in R$
7: **end for**
Algorithm 3 has built the set R, the sub-system/functional group results for each test case.

Algorithm 4 Find Common Symptoms: $R \mapsto SP$

1: Let sp_l be a set of 'test cases results' where l is an index set L
2: Let SP be a set whose members are sets sp_l { SP is the set of 'fault symptoms' for the sub-system}
3: **for all** $r_j \in R$ **do**
4: $sp_l \in \mathcal{P}R \wedge sp_l \in SP$
5: $sp_l \in \bigcap R \wedge sp_l \in SP$ { Collect common symptoms. Analyse the sub-system's fault behaviour under the failure modes in tc_j and determine the symptoms sp_l that it causes in the functional group FG}
Ensure: $\forall a \in sp_l | \forall sp_i \in \bigcap_{i=1..L} SP(sp_i = sp_l \implies a \in sp_i)$
{ Ensure that the elements in each sp_l are not present in any other sp_l set }
6: **end for**
7: The Set SP can now be considered to be the set of fault modes for the sub-system that FG represents Algorithm 4 raises the failure mode abstraction level. The failures have now been considered not from the component level, but from the sub-system or functional group level. We now have a set SP of the symptoms of failure.

Algorithm 5 Treat the symptoms as failure modes of the Sub-System: $SP \mapsto SS$

1: Let SS be a set of faliure modes with failure modes f indexed by l
2: **for all** $sp_l \in SP$ **do**
3: $f_l = ConvertToFaultMode(sp_l)$
4: $f_l \in SS$
5: **end for**
Algorithm 5 is the final stage in the process. We now have a sub-system SS, which has its own set of failure modes. This can now be treated as a component, and used to form functional groups at a higher level of failure mode abstraction.

9.6 To conclude

The technique provides a methodology for bottom-up analysis of the fault behaviour of complex safety critical systems.

9.6.1 Hierarchical Simplification

Because symptom abstraction collects fault modes, the number of faults to handle decreases as the hierarchy progresses upwards. This is seen in real life Systems. At the highest levels the number of faults reduces. A Sound system might have, for instance only four faults at its highest or System level,

$$SoundSystemFaults = \{TUNER\_FAULT, CD\_FAULT, SOUND\_OUT\_FAULT, IPOD\_FAULT\}$$

The number of causes for any of these faults is very large ! It does not matter which combination of causes caused the fault to the user. But as the hierarchy goes up in abstraction level the number of faults goes down.

9.6.2 Tracable Fault Modes

Because the fault modes are determined from the bottom-up, the causes for all high level faults naturally form trees. Minimal cut sets [**?**] can be determined from these, and by analysing the statistical likely hood of the component failures the MTTF and SIL[**?**] levels can be automatically calculated.

Chapter 10

Failure Mode Modular De-Composition

Chapter 11

A Formal Description of FMMD

Chapter 12

FMMD component to module level example : Simple 'ON OFF' Switch

Chapter 13

FMMD component to module level example : Safety Critical 'ON OFF' Switch

Chapter 14

FMMD component to module level example : Reading 4 to 20 mA inputs

Chapter 15

FMMD component to module level example : Thermocouple Input

Chapter 16

FMMD component to module level example : Triac Outputs

Chapter 17

A complete system example, A Safety critical P.I.D temperature controller

Safety critical in that it must not overheat, and that it must alarm for incorrect temperature.

Chapter 18

FMMD tool : Design Issues

reference the MSC document and describe the Java extension classes. Software documentation for fmmd tool.

Chapter 19

Algorithms and Mathematical Relationships Discovered

Chapter 20

A detailed look at the safety systems required by industrial burner controller

Chapter 21

Conclusion

Bibliography

[1] R. P. Feynman. *What do you care what other people think: Harper Collins : ISBN 0-586-21855-6*. harpercollins, 1988.

[2] E. N. Standard. Functional safety of electrical/electronic/programmable electronic safety related systems. EN61508, 2002.

[3] E. N. Standard. Gas burner controllers with forced draft. EN298, 2003.