SRI International

Technical Report SRI-CSL-09-02 • May 2009
Updated January 2010

Reasoning about the Reliability Of Diverse Two-Channel Systems In which One Channel is “Possibly Perfect”

Bev Littlewood
Centre for Software Reliability
City University, London EC1V 0HB, UK

John Rushby
Computer Science Laboratory
SRI International, Menlo Park CA 94025, USA

John Rushby’s research was partially supported by NASA cooperative agreements NNX08AC64A and NNX08AY53A, and by National Science Foundation grant CNS-0720908.

Computer Science Laboratory • 333 Ravenswood Ave. • Menlo Park, CA 94025 • (650) 326-6200 • Facsimile: (650) 859-2844

This document is formatted for two-sided printing with a binding: even-numbered pages should appear on the left and odd-numbered pages on the right when opened as a double-page spread. This page is the back of the title page.

Abstract

This report refines and extends an earlier paper by the first author [25]. It considers the problem of reasoning about the reliability of fault-tolerant systems with two "channels" (i.e., components) of which one, A, because it is conventionally engineered and presumed to contain faults, supports only a claim of reliability, while the other, B, by virtue of extreme simplicity and extensive analysis, supports a plausible claim of "perfection."

We begin with the case where either channel can bring the system to a safe state. The reasoning about system probability of failure on demand (*pfd*) is divided into two steps. The first concerns *aleatory uncertainty* about (i) whether channel A will fail on a randomly selected demand and (ii) whether channel B is imperfect. It is shown that, conditional upon knowing p_A (the probability that A fails on a randomly selected demand) and p_B (the probability that channel B is imperfect), a conservative bound on the probability that the *system* fails on a randomly selected demand is simply $p_A \times p_B$. That is, there is conditional independence between the events "A fails" and "B is imperfect." The second step of the reasoning involves *epistemic uncertainty* represented by an assessor's beliefs about the distribution of $(p_A,\ p_B)$ and it is here that dependence may arise. However, we show that under quite plausible assumptions, a conservative bound on system *pfd* can be constructed from point estimates for just three parameters. We discuss the feasibility of establishing credible estimates for these parameters.

We extend our analysis from faults of omission to those of commission, and then combine these to yield an analysis for monitored architectures of a kind proposed for aircraft.

Contents

Chapter 1

Background

This report is about the assessment of dependability for two-channel design-diverse fault tolerant software-based systems. This type of system architecture is mainly used to provide protection against design faults in safety-critical applications. The report clarifies and greatly extends a note by one of the authors from a few years ago on a specific instance of the topic [25].

The use of intellectual diversity to improve the dependability of processes is ubiquitous in human activity: witness the old saws "Two heads are better than one" and "Belt *and* braces[1]." We all believe that having a colleague review our work—"two heads"—is better than conducting the review ourselves, and we generally consider it prudent to have a backup for complicated machines or plans.

The use of diversity to build reliable systems is widespread in non-software based engineering. For example, in the nuclear industry it is common to have multi-channel protection systems that are based on physically different process variables (e.g., temperature, pressure, flow-rates). The simple idea underlying these applications of diversity is that, whilst it is inevitable that humans make mistakes in designing and building systems, and that these mistakes can cause failures during operation, forcing the systems to be built differently might make the *failure processes* of different versions be diverse also. In other words, you might expect that if channel A fails on a particular demand, there is a good chance that channel B will not fail.

These ideas have been applied to software systems for nearly thirty years, but there has been some controversy about the benefits that the approach brings. On the one hand, there are reports of successful industrial applications of software design diversity: for example, the safety-critical flight control systems of the Airbus fleets [45] have now experienced massive operational exposure [7] with apparently no critical failure. On the other hand, successive carefully conducted experiments—see e.g., [13, 23]—have shown that multiple diverse channels cannot be assumed to fail independently. Thus, if the two channels of a system each have a probability of failure on demand (*pfd*) of 10^{-3}, it would be wrong to claim a *pfd* of 10^{-6} for the system. In fact, there will generally be positive dependence of failures between the two channels, and so the *pfd* will be greater than 10^{-6}.

[1] Or suspenders for North American readers.

Some insight into these experimental results came from probabilistic models of diversity, [14, 24]. This work gave careful attention to the different notions of "independence" that had hitherto been treated somewhat informally, and showed why versions that had been *developed* "independently" could not be expected to *fail* "independently." The key idea in this work is a notion of "difficulty" variation over the demand space—informally, some demands are intrinsically more difficult to handle correctly than others. So even if you knew the marginal probabilities of failure, p_A and p_B, of the two channels (e.g., from extensive operational testing), you could not claim that the probability of failure of *both* channels on a randomly selected demand was $p_A \times p_B$. Instead, when you observe channel A failing on the randomly selected demand, this strengthens your belief that this demand is one of the "difficult" ones, and thus you believe that it is more likely that B will fail: that is, the chance of failure is *greater* than the marginal probability, p_B.

The picture that emerged from this evidence—industrial experience, experiments, and theoretical modeling—has sometimes been taken to undermine claims for the efficacy of software diversity. Whilst the industrial evidence is positive, it inevitably emerges only after years of operating experience, and even then does not easily allow quantitative claims for achieved reliability. The experimental and modeling evidence, on the other hand, has been taken to be negative.

In fact, this negative view seems somewhat unfair. Although the experiments and models show that assumptions of independence of failures cannot be trusted, this does not mean that there cannot be useful benefits to be gained from design diversity. Ruling out independence means that we cannot simply multiply the *pfd*s of the separate channels to obtain the system *pfd*, but there might still be considerable reliability benefits. The modeling results of Littlewood and Miller [24] actually show that there *will* be benefits under quite plausible assumptions, but they do not help quantify these benefits for particular instances; the experiments of Knight and Leveson [22] show that the benefits can be considerable *on average*, but again their results do not help to assess a particular system's reliability.

The position for several years, then, has been this: there is ample evidence that design diversity can "work," in some average sense, in delivering improved reliability, but there is very little evidence as to *how much* improvement it delivers even on average,[2] and the important problem of how to estimate the reliability of a *particular* design-diverse system remains very difficult.

This report concerns this last problem, and it begins with a rather special class of architectures that appear to bring a simplification to the assessment problem. This work was prompted by one author's involvement in the early 1990s with the difficulties of assessing the reliability of the protection system of the UK's Sizewell B nuclear plant. This system comprises the Primary Protection System (PPS), which is software-based, and the

[2]This notion of "on average" efficacy is similar to that used to support claims for other software engineering processes. We know that certain software engineering processes—e.g., different kinds of testing or static analysis [26]—are effective in improving the resulting reliability of the systems to which they are applied. But we usually know very little about *how much* they deliver even on average, and generally nothing at all about what contribution they make to a *particular* system's development (because the particular can be very different from the average). This means that knowledge about the software engineering processes used to build a system is of limited use in estimating its operational reliability.

Secondary Protection System (SPS), which is hard-wired, in a "one-out-of-two" (sometimes written 1oo2) configuration—that is, if either of the systems requests it, the reactor "trips" and enters a safe state.

The PPS provides quite extensive and desirable functionality, over and above that required for a simple reactor trip; for example, it has extensive built-in hardware self-testing and redundancy management features to maximize its availability. The result is that, for a safety-critical system, it is quite complex with about 100,000 lines of code. For this reason it was not thought plausible to make claims for perfection of the software in the PPS; instead, it was required to have a *pfd* no worse than 10^{-3}.

The SPS, on the other hand, has much more restricted functionality, and as a consequence is much simpler (and does not depend upon software for its correct operation). Because of this simplicity of design, it might have been possible to claim that the SPS is free of design faults, and thus could be deemed free of failures in operation from such causes.[3] Of course, a regulator might have justifiable skepticism when faced with such a claim of "perfection": he might be prepared to accept a high confidence in perfection, given extensive suitable supporting evidence, but not accept *certainty*. It might therefore be plausible to contemplate attaching a probability to the claim of perfection for the SPS and then to combine this in some way with the reliability claimed for the PPS to yield a quantifiable claim for the overall system.

Asymmetric fault tolerant architectures (i.e., those in which one channel is claimed to be reliable and the other is possibly perfect) are intuitively plausible for other safety critical systems in addition to nuclear shutdown. For example, in some military aircraft, an extensively functional primary system is used for normal operation, backed up by a "get you home" system of much more restricted functionality; similarly, some heart pacemakers and defibrillators have a purely hardware "safety core" that provides backup to a more complex firmware-driven primary system. Whilst these approaches are recognized as attractive ways for *achieving* reliability and safety, it is less widely appreciated that the asymmetry of the architecture can also aid in *assuring* that the system is sufficiently reliable (or safe), if we can exploit the possibility of perfection for the second channel.

In Chapter 2 of the report this idea will be examined in some detail. For clarity, the account will be conducted in terms of a 1oo2 demand-based system, such as the Sizewell protection system, but (as seems likely now, even for these nuclear systems) both channels will be assumed to be software-based. In Chapter 3 we discuss the application of the ideas in system safety assessment—in particular how the probabilities in the model might be obtained for real systems. Chapter 4 considers Type 2 failures (or uncommanded activations), and Chapter 5 combines the previous results and applies them to "monitored architectures" in which the behavior of a complex operational channel is monitored by a simpler channel that can trigger higher-level fault recovery when safety properties are violated. Architectures of this kind are advocated for "Integrated Vehicle Health Monitoring" (IVHM) in airplanes, and specifically for "Software IVHM." Chapter 6 presents our conclusions.

[3] Of course, the SPS could fail as a result of hardware component failures—as could the PPS. But there are good reasons to believe that such contributions to unreliability are better understood and are small compared to those arising from design faults. For simplicity here and later in this report we shall ignore hardware failures, but it is easy to extend the modeling to include them.

Chapter 2

Reasoning About a One-Out-Of-Two System with a Possibly-Perfect Channel

The first part of the reasoning here is similar to that used in the earlier paper [25]. The main novelty is that the present report establishes *conditional independence* of failure of one channel and imperfection of the other[1] and, in the second part of the chapter, introduces a proper modeling of the epistemic uncertainty about the parameters of the original work. In giving presentations of this work, we have found that some are surprised by, and others are skeptical of our results; we attempt to allay concern by presenting our arguments in some detail.

Our object of study is a 1oo2 demand-based system comprising two channels, A and B. Channel A is assumed to have sufficient functionality, and thus complexity, for it to be infeasible to claim it to be fault-free. We thus have to assume that in operation it will eventually fail, and the criterion of its acceptability (when judged alone) will be that it fails infrequently—that is, its probability of failure on a randomly selected demand, pfd_A, is acceptably small. The kinds of evidence that could be used by an assessor to gain confidence that pfd_A is small include: seeing lots of operational testing without failure, high quality of developers and development process, no faults found in extensive static analysis, etc.

Channel B, on the other hand, has been kept very simple (its functionality is limited and its implementation straightforward) and thus is open to a claim of "perfection."[2] There is a probability, pnp_B, that this channel is *not* perfect. Evidence allowing an assessor to have confidence this probability is small could include: a formal specification and high confidence

[1] Two events X and Y are conditionally independent given some third event Z when $P(X \text{ and } Y \mid Z) = P(X \mid Z) \times P(Y \mid Z)$; we say the independence of X and Y is conditional on Z. Z may be the event that some parameter takes a known value. Unconditional independence is $P(X \text{ and } Y) = P(X) \times P(Y)$ and is not, in general, implied by conditional independence.

[2] This word has been chosen deliberately for its convenience, but in the knowledge that it comes with extensive baggage. In this context it means just that this channel of software will not fail however much operational exposure it receives; it is "fault-free" if we assume that failures can and will occur if and only if faults are present.

that this really does accurately capture the engineering requirements, formal proof that the software is a correct implementation of this specification, absence of failures in test, etc.

There are two kinds of uncertainty involved in reasoning about the reliability of the overall system. The first is *aleatory uncertainty*, or "uncertainty in the world," the second is *epistemic uncertainty*, or "uncertainty about the world" [35]. Aleatory uncertainty can be thought of as "natural" uncertainty that is *irreducible*. For example, if you were to toss a coin you would not be able to predict with certainty whether it would fall as "heads" or as "tails." This uncertainty cannot be eliminated, and any predictions about future tosses of the coin can only be expressed as probabilities. This is true even when we know the probability of heads, which we denote p_H, on a single toss of the coin, such as when the coin is "fair" and $p_H = 0.5$. Real coins, of course, are not fair, so there will be uncertainty about the value of the parameter p_H. This is *epistemic* uncertainty. This uncertainty is reducible: in the case of the coin, we could toss it very many times and observe the frequency of heads in this sequence of tosses. This frequency is an estimate of p_H and the estimate gets closer to the true value of p_H as the number of observed tosses increases (there are limit theorems in probability that detail the nature of this convergence), so the uncertainty reduces.

In much scientific modeling, the aleatory uncertainty is captured conditionally in a model with unknown parameters, and the epistemic uncertainty centers upon the values of these parameters—as in the simple example of coin tossing.[3] For our 1oo2 system the two probabilities, pfd_A and pnp_B, are parameters that capture two types of aleatory uncertainty. For a randomly selected demand, Channel A either does, or does not, fail; pfd_A is just the chance that it *does* fail. As in the case of a coin toss, this chance can be given a classical frequentist interpretation, as the proportion of failures in a hypothetical infinitely large sequence of independently selected demands.

Similarly, Channel B either is, or is not, perfect; pnp_B is just the chance that it is *not* perfect. The way to interpret this is to think of all the programs that *might* have been developed, by comparable engineering processes, to solve the problem of Channel B, in the spirit of [14, 24], and interpret pnp_B as the probability of a randomly selected program not being perfect. This probability will be partly dependent on the nature of the problem being solved—for a "hard" problem it might be expected that the probability is greater than for an "easier" problem—and partly on the quality of the software engineering and assurance methods employed.

A result that we shall prove in Section 2.1 is that these two parameters are sufficient to describe the uncertainty about the *system* probability of failure. Specifically, conditional on knowing the actual values p_A and p_B of pfd_A and pnp_B, respectively, the system probability of failure is no worse than $p_A \times p_B$. The epistemic uncertainty here solely concerns the values of p_A and p_B: an assessor will not know these values with certainty. His beliefs about these values—and here we adopt a Bayesian subjective interpretation of probabilities—will be represented by a joint distribution

$$F(p_A, p_B) = P(pfd_A < p_A, pnp_B < p_B) \tag{2.1}$$

[3]The situation can be more complicated than this, of course. There may be uncertainty about which of several models is "correct," and this uncertainty may not be captured by a parametric model.

and his probability for system failure will then be

$$\int_{\substack{0\le p_A\le 1\\ 0\le p_B\le 1}} p_A \times p_B \, dF(p_A, p_B). \tag{2.2}$$

Generally, the assessor's beliefs about (p_A, p_B) will not be independent: that is, the bivariate distribution (2.1) does not factorize into a product of the marginal distributions of the random variables pfd_A and pnp_B and so it is not straightforward to derive a numerical value for (2.2). This is an important issue which is examined in Section 2.2

Our reasoning about the reliability of the system now proceeds in two stages, the first involving the aleatory uncertainty, the second the epistemic uncertainty.

2.1 Aleatory Uncertainty

We shall assume in what follows that our interest centers upon the system reliability expressed as the probability of its failure upon a (operationally) randomly selected demand. Note that other things might be of interest, for instance the probability of surviving a specified number of demands (e.g., the number expected to be encountered during a specific mission, or during the lifetime of the system), but we shall not consider these here.

For aleatory uncertainty we have, by the formula for total probability, the following conditional *pfd*.

$$\begin{aligned}
&P(\text{system fails on randomly selected demand} \mid pfd_A = p_A, pnp_B = p_B) \qquad (2.3)\\
&= P(\text{system fails} \mid A \text{ fails}, B \text{ imperfect}, pfd_A = p_A, pnp_B = p_B)\\
&\quad \times P(A \text{ fails}, B \text{ imperfect} \mid pfd_A = p_A, pnp_B = p_B)\\
&\quad + P(\text{system fails} \mid A \text{ succeeds}, B \text{ imperfect}, pfd_A = p_A, pnp_B = p_B)\\
&\quad \times P(A \text{ succeeds}, B \text{ imperfect} \mid pfd_A = p_A, pnp_B = p_B)\\
&\quad + P(\text{system fails} \mid A \text{ fails}, B \text{ perfect}, pfd_A = p_A, pnp_B = p_B)\\
&\quad \times P(A \text{ fails}, B \text{ perfect} \mid pfd_A = p_A, pnp_B = p_B)\\
&\quad + P(\text{system fails} \mid A \text{ succeeds}, B \text{ perfect}, pfd_A = p_A, pnp_B = p_B)\\
&\quad \times P(A \text{ succeeds}, B \text{ perfect} \mid pfd_A = p_A, pnp_B = p_B).
\end{aligned}$$

The last three terms of the right hand side of (2.3) are zero: the system does not fail if either A succeeds or B is perfect. We shall assume conservatively that if B is not perfect, then it will fail with certainty whenever A fails, as in the original version of this work [25]: B brings no benefit if it is not perfect. This (very) conservative assumption is important in the following reasoning where, in the first term on the right hand side of (2.3), the first factor in the product is replaced by 1, to yield:

$$
\begin{aligned}
&P(\text{system fails on randomly selected demand} \mid pfd_A = p_A,\ pnp_B = p_B) \\
&\quad \leq \quad P(A \text{ fails},\ B \text{ imperfect} \mid pfd_A = p_A,\ pnp_B = p_B) \\
&\quad = \quad P(A \text{ fails} \mid B \text{ imperfect}, pfd_A = p_A,\ pnp_B = p_B) \qquad (2.4) \\
&\qquad\qquad \times P(B \text{ imperfect} \mid pfd_A = p_A,\ pnp_B = p_B).
\end{aligned}
$$

Now, the fact that B is imperfect tells us nothing about whether or not A will fail on *this demand*. So we have

$$
\begin{aligned}
&P(A \text{ fails} \mid B \text{ imperfect}, pfd_A = p_A,\ pnp_B = p_B) \qquad (2.5) \\
&\quad = \quad P(A \text{ fails} \mid pfd_A = p_A,\ pnp_B = p_B) \\
&\quad = \quad P(A \text{ fails} \mid pfd_A = p_A)
\end{aligned}
$$

and so, substituting (2.5) in (2.4),

$$
\begin{aligned}
&P(\text{system fails on randomly selected demand} \mid pfd_A = p_A,\ pnp_B = p_B) \\
&\quad \leq \quad P(A \text{ fails} \mid B \text{ imperfect}, pfd_A = p_A,\ pnp_B = p_B) \\
&\qquad\qquad \times P(B \text{ imperfect} \mid pfd_A = p_A,\ pnp_B = p_B) \\
&\quad = \quad P(A \text{ fails} \mid pfd_A = p_A) \times P(B \text{ imperfect} \mid pnp_B = p_B) \\
&\quad = \quad p_A \times p_B. \qquad (2.6)
\end{aligned}
$$

This argument follows that in [25] quite closely; the difference is that here we show that there is *conditional independence* between the events "A fails" and "B imperfect" when pfd_A and pnp_B are known.

The importance of this independence result can be seen by contrasting the situation here with the more usual 1oo2 system, where both channels are sufficiently complex that neither is open to a believable claim of perfection. In the latter case, we cannot apply the reasoning in step (2.5) and so it is not possible to claim independence of failures in the two channels, *even conditionally* given pfd_A and pfd_B (in an obvious notation). Instead, even at this stage of reasoning where we are concerned only with aleatory uncertainty, we need to deal with dependence of failures. Thus, it is shown in [24], for example, that the conditional probability of such a system failing on a randomly selected demand is

$$pfd_A \times pfd_B + Cov(\theta_A, \theta_B)$$

where θ_A, θ_B are the difficulty function random variables for the two channels. So an assessor's beliefs about pfd_A and pfd_B are not alone sufficient for him to reason about the reliability of the system: he also needs to know the covariance of the difficulty functions. And this is before we consider the problem of epistemic uncertainty concerning parameters such as pfd_A, pfd_B, and $Cov(\theta_A, \theta_B)$, which would involve further dependencies.

In contrast, (2.4–2.6) above show that knowledge of the values of pfd_A and pnp_B are sufficient for a complete description of aleatory uncertainty via the (conservative) conditional independence expression for the system pfd, (2.6).

Lest it seem that a particular choice was needed at equation (2.4), it is worth observing that this could equally have been written as

$$\begin{aligned} & P(\text{system fails on randomly selected demand} \mid pfd_A = p_A,\ pnp_B = p_B) \\ & \quad \leq \ P(A \text{ fails},\ B \text{ imperfect} \mid pfd_A = p_A,\ pnp_B = p_B) \\ & \quad = \ P(B \text{ imperfect} \mid A \text{ fails}, pfd_A = p_A,\ pnp_B = p_B) \\ & \qquad \times P(A \text{ fails} \mid pfd_A = p_A,\ pnp_B = p_B), \end{aligned} \tag{2.7}$$

where the conditioning is reversed in (2.7).

Here, the failure of A on this demand tells us nothing about the imperfection of B, since this is a global property with known probability p_B, and so we have

$$\begin{aligned} & P(\text{system fails on randomly selected demand} \mid pfd_A = p_A,\ pnp_B = p_B) \\ & \quad \leq \ P(B \text{ imperfect} \mid A \text{ fails}, pfd_A = p_A,\ pnp_B = p_B) \\ & \qquad \times P(A \text{ fails} \mid pfd_A = p_A,\ pnp_B = p_B) \\ & \quad = \ P(B \text{ imperfect} \mid pnp_B = p_B) \times P(A \text{ fails} \mid pfd_A = p_A) \\ & \quad = \ p_B \times p_A \end{aligned}$$

as before.

The attributes of "perfection" that enable the reasoning employed at (2.4) and (2.7) are: first, that its satisfaction guarantees its channel does not fail on a given demand (this is needed to eliminate the last two terms in (2.3)); second, that it is a global property (i.e., independent of *this* demand, which is needed in (2.4) and (2.7)); and third, that it is aleatory (i.e., its uncertainty is irreducible).

Other global properties, such as "has a valid proof of correctness" (whose negation would be "has a flawed proof of correctness"), appear plausible but are not aleatory. Proof of correctness is relative to a specification, and it is possible this specification is flawed. Thus, absence of failure on a given demand (needed for the first point above) is contingent on correctness of the specification, which is an additional parameter and a reducible source of uncertainty.

Hence "perfection" as we have defined it is *precisely* the property required for this derivation. Properties such as "has a valid proof of correctness" are important in assigning an epistemic value to the probability of perfection and are discussed in Chapter 3. First, though, we examine the basic problem of epistemic uncertainty about the model parameters.

2.2 Epistemic Uncertainty and the Structure of $F(p_A, p_B)$

The previous section has derived a conservative bound for the probability of failure on demand for a 1oo2 system, conditional on knowledge of the values of the random variables pfd_A and pnp_B. In practice, of course, pfd_A and pnp_B will not be known with certainty, and this is where the epistemic uncertainty comes in, via the assessor's posterior beliefs represented by the distribution, (2.1). "Posterior" here means that this distribution takes account of all evidence the assessor has acquired about the unknown parameters.

Practical interest centers upon the *unconditional* probability of system failure on a randomly selected demand. By the formula for total probability, and using the Riemann-Stieltjes integral, this is

$$\begin{aligned} &P(\text{system fails on randomly selected demand}) \\ &= \int_{\substack{0\le p_A\le 1\\ 0\le p_B\le 1}} P(\text{system fails on randomly selected demand} \,|\, pfd_A = p_A,\, pnp_B = p_B)\, dF(p_A,\, p_B) \end{aligned}$$

so, substituting (2.6),

$$\le \int_{\substack{0\le p_A\le 1\\ 0\le p_B\le 1}} p_A \times p_B \, dF(p_A,\, p_B), \tag{2.8}$$

which provides the formal derivation for (2.1).

This two-stage reasoning—aleatory uncertainty followed by epistemic uncertainty—makes clear the source of any dependence here: it can arise only from the epistemic uncertainty, via the posterior distribution $F(p_A,\, p_B)$. Even though the events "A fails" and "B is imperfect" are conditionally independent at the aleatory level, an assessor's beliefs about the probabilities of these events may not be independent. If they are independent, so that $F(p_A,\, p_B)$ factorizes as $F(p_A) \times F(p_B)$, then

$$\begin{aligned} &P(\text{system fails on randomly selected demand}) \\ &\le \int_{\substack{0\le p_A\le 1\\ 0\le p_B\le 1}} p_A \times p_B \, dF(p_A,\, p_B) \qquad (2.9) \\ &= \int_{0\le p_A\le 1} p_A \, dF(p_A) \times \int_{0\le p_B\le 1} p_B \, dF(p_B) \qquad (2.10) \\ &= P(A \text{ fails}) \times P(B \text{ imperfect}) \\ &= P_A \times P_B. \end{aligned}$$

That is, if the assessor's beliefs about pfd_A and pnp_B are independent, then the bound on the probability of system failure is just the product of the assessor's posterior probabilities of the events "A fails" and "B imperfect" (i.e., the means of the distributions for pfd_A and pnp_B, respectively, taking account of any experimental and analytical evidence that may be available). In this case the reliability assessment is particularly simple—it involves only the assessment of two parameters.

If, on the other hand, there is positive association between an assessor's beliefs about pfd_A and pnp_B, (2.9) will be larger than (2.10). Readers may think that this is the situation that will apply in most real-life cases. An assessor then is faced with expressing his beliefs as a complete bivariate distribution. Elicitation of experts' subjective probabilities is an active research area in Bayesian statistics, and there have been considerable advances in recent years in techniques and tools: see [36] for a good introduction to this work. We shall not consider this general elicitation problem further here, since for a particular system

the elicitation process will depend upon the exact details of the application. Instead, we consider a simplification that will ease this elicitation task in general. Our approach is for the assessor to capture the dependency in his beliefs about the parameters in a conservative way, by lumping all the dependency into a third parameter.

Consider the ways in which dependency might arise. There seem to be only two possibilities: one is intrinsic difficulty of some demands, the other is common cause errors. We consider these in turn.

The assessor might observe that some parts of the assurance case for the possibly perfect channel B are harder than others. But should this affect his confidence in corresponding parts of the reliable channel A and his beliefs about its probability of failure? We argue not: it is accepted that the reliable channel fails with some probability, and we have an assessment of that probability (e.g., by statistically valid random testing).

Dually, the assessor might observe in testing that the A channel fails more often on some classes of demands than others. But should this affect his beliefs about the probability of perfection of the B channel? We discuss how probability of perfection can be estimated in Chapter 3, but note here that assurance techniques that can support claims of perfection, such as mechanically checked proofs of correctness, are "uniformly credible" across all cases. By this we mean that the case analysis induced by the structure of the software and by the formal specification against which it is proven correct may yield some cases that are harder to prove (i.e., require more work) than others, but this will not affect our confidence in the correctness (i.e., the credibility) of those cases: a (mechanically checked) proof is a proof.

This claim notwithstanding, our focus is *possibly* perfect systems and we do accept that proofs can be flawed. However, for the reasons just described, we think that failures of Channel A and flaws in the proof of B are unlikely to concern the same demands unless the root cause is in a higher level that is common to both channels, such as the interpretation of requirements.

Thus, faults common to both channels, such as high level misunderstanding of the system requirements, seem the only way in which dependency might arise between the assessor's beliefs about p_A and p_B. One way forward, therefore, is for the assessor to place some probability mass, say C, at the point (1, 1) in the (p_A, p_B)-plane to represent his subjective probability that there are such common faults. The effect of this is that the assessor believes that if there *are* such faults, then A will fail with certainty ($p_A = 1$), B will be imperfect with certainty ($p_B = 1$), and hence $p_A \times p_B = 1$. In other words, there is a probability C that the system is certain to fail on a randomly selected demand (since we assume that if B is imperfect, it always fails when A does). This is clearly conservative, and may be very much so.

The assessor then has the following conservative unconditional probability of failure of the system on a randomly selected demand (again by a version of the formula for total

probability).

$$
\begin{aligned}
&P(\text{system fails on randomly selected demand}) \\
&\leq \int\limits_{\substack{0\leq p_A\leq 1\\ 0\leq p_B\leq 1}} p_A \times p_B \, dF(p_A, p_B) \\
&= C \times \int p_A \times p_B \, dF(p_A, p_B \,|\, p_A = p_B = 1) \\
&\quad + (1-C) \times \int\limits_{\substack{0\leq p_A< 1\\ 0\leq p_B< 1}} p_A \times p_B \, dF(p_A, p_B \,|\, p_A \neq 1, p_B \neq 1) \\
&= C + (1-C) \times \int\limits_{\substack{0\leq p_A< 1\\ 0\leq p_B< 1}} p_A \times p_B \, dF(p_A, p_B). \qquad (2.11)
\end{aligned}
$$

If the assessor believes that all dependence between his beliefs about the model parameters has been captured conservatively in C, the conditional distribution in the second term of (2.11) now factorizes, and we have:

$$
\begin{aligned}
&P(\text{system fails on randomly selected demand}) \\
&\leq C + (1-C) \times \int\limits_{0\leq p_A< 1} p_A \, dF(p_A) \times \int\limits_{0\leq p_B< 1} p_B \, dF(p_B) \\
&= C + (1-C) \times P_A^* \times P_B^* \qquad (2.12)
\end{aligned}
$$

where P_A^* and P_B^* are the means of the posterior distributions representing the assessor's beliefs about the two parameters, each conditional on the parameter not being equal to 1 (i.e., A not certain to fail, B not certain to be imperfect).

If C is small (as will be likely in the contexts in which such reasoning takes place in real life), we can replace the factor $(1-C)$ in (2.12) by 1, and we can substitute P_A, the unconditional probability of failure of the A channel, for P_A^*, and P_B, the unconditional probability of imperfection of the B channel, for P_B^*, to yield the *conservative* approximation

$$C + P_A \times P_B. \qquad (2.13)$$

Assessors may find it easier to formulate their beliefs about these unconditional probabilities. The same simplification can be applied to similar formulas developed in later chapters and the resulting approximations will all be conservative.

Chapter 3

Estimation of Model Parameters

In this chapter, we expand the discussion of epistemic estimation and describe in more detail the surrounding context of system safety assessment and the development of evidence and probabilities for possible perfection. We use the word "estimation" in this chapter for the process by which an assessor expresses his "expert beliefs" in terms of numeric values for the parameters that appear in Formula (2.12) or its simplification (2.13). We do not mean to imply that these numbers have aleatoric, *in*-the-world meaning, of course. Rather, they are representations of the assessor's beliefs *about* the world.

The decision to deploy a safety-critical or other kind of critical system is based on careful assessment of risks, estimating both the likelihood of undesired events, and their potential costs and consequences in terms of environmental harm, loss of life, erosion of national security, loss of commercial confidence or reputation, and so on. The assessment is highly specific to the system concerned and its context, and nowadays is generally grounded in an argument-based safety or assurance case (see, for example [3, 41, 51, 57, 59]). Such a case begins with *claims* that enumerate the undesired loss events and the tolerable failure rate or risk associated with them (e.g., "as low as reasonably practicable" (ALARP) [58], or "so unlikely that they are not anticipated to occur during the entire operational life of all airplanes of one type" [15, paragraph 9.e(3)]). *Evidence* about the system and its processes of construction are developed, and an *argument* is constructed to justify satisfaction of the claims, based on the evidence. This process may recurse through subsystems, with substantiated claims about a subsystem being used as evidence in a parent case.

The upper levels of an assurance case will often decompose into subcases based on enumeration of hazards, or failure modes, or architectural components. In the case of multi-channel systems, there will be upper-level analysis of the mechanism (if any) that coordinates them (e.g., a voter or arbiter), and of causes or modes of failure that may be common to all channels. Common causes may include misunderstanding of the overall system requirements, shared assumptions, and shared or common mechanisms. Much engineering expertise and many associated analysis tools are available for identifying shared or common mechanisms: for example, common cause analysis, common mode analysis, zonal safety analysis, etc. There is rather less expertise in identifying shared assumptions: for example, it is often implicitly assumed that if a single source fans out to provide inputs

to several channels, then all channels that obtain valid inputs will agree on the values of those inputs. This assumption is not only implicit, but it is generally unjustified, and often wrong: "Slightly Out of Specification" (SOS) faults are one mechanism that can lead to "Byzantine" violations of source congruence [5, 12].

One way to identify assumptions is through mechanically assisted formal verification: this is the process of proving that a formal model of a component or mechanism, in the context of a modeled environment, correctly delivers specified properties. Assumptions are generally represented in the model of the environment (e.g., as axioms), and the proof will not succeed until these are identified and formulated adequately. The benefit of formal verification in this context is that it forces consideration of *all possible* cases induced by the specification and models, something that is seldom achieved by unassisted human review, or by testing, thereby facilitating discovery of assumptions. Furthermore, formal verification provides *mechanized* assistance in the consideration of all possible cases through use of symbolic analysis (e.g., by considering the behavior of a program on symbolic inputs such as x and y, rather than concrete values such as 23.46 and 17.3 [48]), further facilitating the efficient and reliable discovery and formulation of assumptions. Once assumptions have been identified, through these or other means, human expertise can be used to assess their plausibility, significance, and commonality across different channels.

A related source of shared errors is a *lack* of common assumptions and expectations across subsystems: this is a leading cause of the "integration explosion" that often occurs when subsystems are assembled for the first time. In these cases, subsystems seem correct in isolation, but fail in combination. One way to alleviate this problem is, again, through formal verification: given formal models of subsystem and component requirements and properties, mechanically assisted verification can explore their behavior in composition, forcing consideration of all possible interactions and often revealing overlooked cases, mismatched assumptions, and undesired emergent behavior.

This approach also can be used to analyze correctness of the top-level requirements: we use formal verification to show that these requirements, in the context of a model of the system environment, deliver desired properties (see [49] for an elementary example).

By these and other means, the arguments and supporting evidence for the correctness of top-level requirements and for the absence of common-cause errors in assumptions, mechanisms, and failure modes across separate channels can be developed in great detail and subjected to substantial analysis and empirical investigation. This is necessary because any flaws in these elements will dominate overall system reliability and also its assessment, no matter how this is performed. We can then propose that an assessor's subjective probability for system failure on a randomly selected demand will take the form $C + (1 - C) \times D$, where C is due to residual doubts about the top-level issues of system requirements and hazards common to all channels, and D is due to the specific architecture of the system and its constituent channels.

The probability C must be small if the system is to be fit for a critical purpose, but it may not be realistic or useful to strive to make it arbitrarily small. In the UK, for example, it is common for assessors to impose a *claim limit* of around 10^{-4} or 10^{-5} for the probabilistic assessment of any system [56]; this can be interpreted as a judgement that they will always retain residual doubts about the top-level issues and that claims for these,

expressed here as C, should be taken only so far, and no further. In UK civil nuclear systems, there is a specific recommended limit of 10^{-5} for probabilistic assessment of common-cause faults [57, paragraph 172].

Thus, in any multi-channel system the assessor must, on the one hand, be provided with sufficiently strong evidence and credible arguments for correctness of the top level requirements and absence of hazards common to all channels that the quantity represented by C is small enough to be useful. But on the other hand, claim limits or other expressions of caution ensure that this quantity is sufficiently conservative that it really can absorb the hazards common to all channels and thus the assessor's beliefs about the probabilities for the separate channels can be treated as independent. In other words, claim limits prevent the value of C being pared away to the point where it threatens this independence.

Thus, we argue, an assessor's conservative probability of failure of the form $C + (1 - C) \times D$, with C around 10^{-5} is entirely reasonable. For a 1oo2 architecture with a possibly perfect channel, D will be $P_A^* \times P_B^*$ (from (2.12)), and it remains to consider the processes for obtaining, or eliciting, P_A^* and P_B^*.

The first of these is the assessor's probability of failure on demand for the reliable channel. For values down to about 10^{-4}, it is feasible to provide strong and credible evidence by statistically valid random testing [8, 27]. By statistically valid here we mean that the test case selection probabilities are exactly the same as those that are encountered in real operation, and that the oracle (used to determine whether a demand has been executed correctly or not) operates correctly. See [30] for an example of such testing used to assess the reliability of a reactor protection system. In the event that there is doubt about the representativeness of the test case selection, or of the correctness of the oracle, then this epistemic uncertainty must be incorporated into the assessor's probability P_A^*. An expert assessor might take account of other kinds of evidence as well as that obtained from testing in arriving at his P_A^*. We shall not discuss this issue in detail here, but note that Bayesian Belief Nets (BBNs) are one formalism for handling such disparate sources of evidence and uncertainty (see, for example, [28]).

For the possibly perfect channel, the assessor must express his subjective probability of its imperfection: that is, his degree of belief that there is at least one demand on which it fails. A rational process for developing this probability uses Bayesian methods to modify a prior belief through the acquisition of evidence to yield a refined posterior belief (see [27, sidebar] for a brief description of this process). The strongest evidence will come from scrutiny and analysis of the design and implementation of the channel concerned. As the claim is "perfection," the analysis will likely include extensive testing and formal verification. Notice that if testing reveals a failure, or if formal verification or other analysis reveals a fault, then the claim of perfection is destroyed and the channel must be replaced in its entirety.[1] In the following, therefore, we consider only the case in which testing reveals no failures.

Formal verification may consider the abstract design of the software and its algorithms, or detailed models developed in a model-based design framework, or the executable code, or

[1] We are speaking here of the testing and verification performed for assurance and evaluation; discovery and repair of failures and faults in earlier stages of development may be acceptable in some lifecycle processes.

all of these. In addition, there may be formalizations of the non-software elements that are part of the channel—for example, its sensors and actuators and the physical processes that they monitor and control, the external environment that may interact with the system (e.g., by injecting faults), and any human operators and their mental models [46] and cognitive states. The properties verified may range from simple absence of runtime anomalies, such as those guaranteed by static analysis, to full functional correctness with respect to a detailed requirements specification. The subjective probability of imperfection will obviously be influenced by consideration of how comprehensive is the formal verification and the extent of what has *not* been formally verified or otherwise examined in comparable detail; it will also be tempered by consideration of the complexity of the design and its implementation. For those elements that are formally verified, it is instructive to consider the main hazards to the soundness of this process.

1. The basic requirements or assumptions for the system may have been misunderstood or established incorrectly. This hazard lies outside those addressed by formal verification: it is included in the common-cause probability C.

   However, it does seem that incorrect requirements are the primary source of failure in many safety-critical systems (see, e.g., recent aircraft incident and accident reports [1, 2, 33, 34, 55]), so we need to be sure that a credible value for C is not so large as to dominate all other considerations.

   We anticipate that software that is subjected to formal verification, and which is used in a context for which a probability of perfection is required, will be relatively small and simple and will be for safety backup and monitoring purposes (essentially, to guard against failures of this very type in the primary safety channel or the operational software). The requirements for software such as this are taken directly from the safety case, so any errors here reflect flaws in the safety case and invalidate far more than the software: they call the entire system and its certification into question. Hence, we believe that errors in requirements and assumptions will not dominate assessments of a probability of perfection.

2. The requirements, assumptions, or design may be formalized incorrectly or incompletely. There are three subcases to this concern.

   (a) Elements of the specification may be inconsistent: this renders the specification meaningless and it becomes possible to prove anything.

   Constructive specifications in a suitable specification framework (e.g., one that requires, among other things, demonstration that all recursive functions are well-founded) are *conservative extensions* of their logic and are therefore always consistent (if the base logic is) [53]. ACL2 [21], Coq [10], HOL [17], and Isabelle [39] are examples of verification systems (i.e., theorem provers for logics that are suitable for system specification and verification tasks) that favor constructive specifications. However, constructive specifications are not always appropriate; for example, when specifying assumptions, it is generally more appropriate to express them as axioms (we wish to state the assumptions, not implement them).

In this case, consistency of the axioms may be ensured by showing that they have a model that is conservative (e.g., defined constructively). The PVS verification system [37], for example, supports this kind of demonstration through theory interpretations [38].

(b) Elements of the specification may be just plain wrong: although logically consistent, they do not correctly formalize the requirements, design, or assumptions. This is generally the dominant hazard in formal specification and analysis. Formal specifications that have not been subject to some form of mechanized analysis are no more likely to be correct than programs that have never been run (in fact, less so, since nonspecialists generally have better intuition about programs than they do about formal specifications). When a number of unmechanized specifications in the Z language (including some of industrial significance) were subjected to mechanized analysis, most were found to contain flaws [50].

The most effective ways to ensure that formal specifications capture their intent are to "challenge" them by attempting to prove putative theorems (i.e., "if I've got it right, this ought to follow, but that ought to yield a counterexample"), or to explore the behavior of executable interpretations of the specification: some constructive logics are directly executable, and a large fragment even of PVS's higher order logic with quantifiers can be evaluated efficiently (i.e., at speeds comparable to those of a functional programming language) [11].

Some verification systems, such as PVS, identify all the axioms and definitions on which a formally verified conclusion depends: if these are correct, then logical validity of the verified conclusion follows by soundness of the verification system. Such identification allows particular scrutiny to be applied to these elements. The axioms and definitions that underpin a verification are generally of two kinds: those that are directly concerned with the subject matter of the system (its requirements, design, assumptions, and so on), and those used in developing the various theories that are required to express this subject matter (e.g., the theories of clocks, synchronous systems, ordinal numbers, and so on). Many of the latter theories will be widely used in other formal developments, so the burden of ensuring their correctness is supported by a broadly shared social process.

Even if a theory or specification is formalized incorrectly, it does not necessarily invalidate all theorems that use it: only if the verification actually exploits the incorrectness or inconsistency will the validity of the theorem be in doubt (and even then, it could still be true, but unproven). The second author has performed many formal verifications and flaws in some of his specifications have been identified by others [40], but in no case did these flaws invalidate the theorems claimed.

(c) The formal specification and verification may be discontinuous or incomplete: discontinuities can arise when several analysis tools are applied in the same software development (e.g., a theorem prover, a model checker, a source code static analyzer, and formally-based security and timing analyzers). Concerns are that different tools may ascribe different semantics to the same specification, transla-

tions between notations may introduce flaws, and there may be unintended gaps that allow some aspect to escape analysis. There is no simple resolution to these concerns: combinations of specialized analysis tools are outstripping the capabilities and performance of monolithic tools and seem to represent the future of the field. Integrating frameworks such as an "Evidential Tool Bus" [47] suggest one way forward.

The most significant incompleteness is generally the gap between the most detailed level of specification that is formally analyzed (e.g., algorithms expressed in a functional programming notation, or a model-based design using state machines) and the input (e.g., C code and "make" files) to the software development environment that generates executable code. Manual translation between these notations may introduce faults, and assumptions in the formal development (e.g., interpretation of mathematical functions such as *sqrt*) may be violated by the execution environment (e.g., due to finite precision).

In an ideal world, the execution environment would itself be provided with a formal specification and strong evidence for correctness, such as that delivered by formal verification. Research projects have accomplished impressive feats in formally verifying such "stacks" of software and hardware [31], but most applications today must rely on informal evidence from extensive and widespread use of their execution platforms, buttressed by testing of their specific configurations.

A plausible compromise would make formal analysis as comprehensive as reasonably practicable, but also employ comprehensive testing on the execution platform. Automated generation of test cases is a popular application of formal methods [18, 60] and tests generated from the lowest level formal specification (which also serves as the test oracle) can provide evidence that the execution behavior matches this specification.

Even when formal verification is employed comprehensively, safety-critical software should be thoroughly tested, as this provides an independent "leg" to the assurance case [6] and also probes the assumptions under which the verification was performed. (The "penetration testing" performed on secure systems explicitly targets assumptions used in formal verification, as these are considered the most attractive points of attack.) Automatically-generated tests, derived from a formal specification, can target specific coverage criteria such as MC/DC (Modified Condition/Decision Coverage) [42], which is required for safety critical software in commercial aircraft (i.e., DO-178B Level A [43]). Any failure discovered in final testing (as opposed to exploratory testing undertaken earlier in development) calls the whole system assurance into question.

We note that if the reliable channel A and the possibly perfect channel B both use identical execution environments, then residual concerns about faults in this environment need to be included in the common cause probability C.

3. The theorem prover, model checker, or other mechanized analysis tool may be unsound.

The concern here is that it may prove a false theorem, not that it may fail to prove a true one (that is completeness). Theorem provers are complex programs (generally far more complex and sophisticated than the specifications and programs that they verify), and concern about their soundness is generally the dominant concern for nonspecialists ("who will guard the guardians"?).

There are several ways to mechanize theorem proving in support of verification. In one way, a search is performed to find a rule of inference that will move things forward from the current proof state. The search may be massive, but soundness depends only on correct application of the selected inference rule (which includes checking that it *is* applicable). Some theorem provers (generally referred to as "LCF-style" [16]) are deliberately designed to have a very small core set of fairly primitive rules; larger sets of more powerful rules can be defined in terms of these, but soundness depends only on the kernel code that implements the core rules. The hope is that a small kernel has a high probability of attaining perfection, and may even be proved correct. Indeed, the kernel of HOL Light, which is about 400 lines of OCAML, has been proven sound by a stronger version of itself [19] (by Gödel's second incompleteness theorem, a sound theorem prover cannot prove its own soundness, so strengthening is needed).

An objection to the LCF-style approach is that it is inefficient: the overhead (which can be exponential) of reducing a big proof step down to invocations of many core rules is always present, even when the prover is being used for the purpose of "exploration" in the early stages of proof development, rather than for final assurance. An alternative approach is for an untrusted theorem prover to generate "proof logs" that can be certified by an independent trusted checker (observe that this is a two-channel architecture similar to the one whose reliability is under consideration); the overhead of log generation and checking can be turned off when not required. There is a tension between the size of the proof log and the complexity of the trusted checker: traditionally, the checker has been very simple, and the log correspondingly huge and expensive to generate. Recent work suggests the feasibility of more powerful checkers (in effect, small theorem provers), which are themselves formally verified [52]. These checkers can use much more succinct logs (little more than hints), and confidence in their own verification can be based on one-time use of more primitive checkers or on a diversity of checkers or verifiers. Diverse verifiers are available for certain standard classes of verification problems such as those that can be reduced to SAT and SMT solving [47].

If the assessor can substantiate a claim of 10^{-3} or 10^{-4} for probability of failure of the reliable channel, then a balanced case would require similar probability of imperfection for the possibly perfect channel. If we take 10^{-3} as an example, then we suggest that the bulk of this "budget" should be divided between items 2b and 2c, with a small fraction, say 10^{-4}, allocated to item 3; as explained in the text above, item 1 is included in C, and item 2a can be eliminated.

We believe that through sufficiently careful and comprehensive formal challenges, it is indeed feasible and plausible that an assessor can assign a subjective posterior probability of imperfection on the order of 10^{-3} or 10^{-4} to the formal statements on which a formal ver-

ification depends (item 2b). Through testing and other scrutiny, we believe a similar figure can be assigned to the probability of imperfection due to discontinuities and incompleteness in the formal analysis (item 2c). And, by use of a verification system with a trusted or verified kernel, or trusted, verified, or diverse checkers, we believe an assessor can assign a posterior probability of 10^{-4} or smaller to the concern that the theorem prover and other components of the mechanized formal verification system may have incorrectly verified the theorems that attest to perfection (item 3).

Thus, in summary, we argue that it is plausible for well-designed and highly assured 1oo2 systems to support claims on the order of 10^{-5} for C, the probability of failure due to causes common to both channels, 10^{-3} for P_A^*, the probability of failure by the reliable channel, and 10^{-3} for P_B^*, the probability of imperfection of the possibly perfect channel. Substituting these values in (2.12) we obtain $10^{-5} + (1 - 10^{-5}) \times 10^{-3} \times 10^{-3}$ and thereby substantiate a conservative assessment of about 1.1×10^{-5} for the probability of failure on demand by the overall system. This overall assessment and its constituent quantities are comparable to those that have been employed for real systems, but we believe this is the first account that provides an argument by which they can be rigorously defended.

The relative sizes of the quantities appearing in this assessment are worth some comment. It is feasible that an assessor could assign a value as small as 10^{-4} to P_A^*, and a similar value seems plausible for P_B^*; their product is then 10^{-8}, which is within range of the ultra-high dependability requirements for transportation systems such as trains and aircraft. The fly in the ointment is common-cause failure, where it may be difficult to assess a probability smaller than 10^{-5} to C, and this therefore sets the overall limit on claimed reliability. There are two ways to interpret this observation: one is to conclude that asymmetric architectures and their analysis—or at least our approach to simplifying their problem of epistemic estimation—are inadequate to those applications requiring ultra-high dependability because C is a limiting factor; the other is to conclude that our analysis is cautionary and that requirements and claims for ultra-high dependability should be viewed skeptically, especially if they depend on assigning very low probabilities to common cause events. We note that the Airbus A340 fuel emergency to be discussed in Chapter 5 on Page 26 was essentially a common-cause failure.

We return to this topic in Chapter 5, where we develop an analysis for the kind of monitored architectures proposed for aircraft, but first we need to examine failures of commission.

Chapter 4

Failures of Commission

In this chapter, we turn from failures of omission to those of commission: those that perform an action (such as shutting down a reactor) when it is not required. In some system contexts, this kind of failure can have serious safety consequences, while in others it may be a costly nuisance, and in yet others merely an annoying "false alarm." We study failures of this kind because they are of some importance in themselves, and because they lay the groundwork for our analysis of monitored architectures in Chapter 5.

In a 1oo2 system, either channel can put the system into a safe state. Hence, the failure of one channel to bring the system to a safe state when this should be done (i.e., on a "demand") can be compensated by the other channel working perfectly. But, dually, the 1oo2 arrangement means that a single channel can force the system to a safe state when it is not necessary to do so (i.e., when there is no demand or, as we shall sometimes say, on a "nondemand"). We can call the failure to bring the system to a safe state when it is necessary to do so a "Type 1" failure (a failure of omission), and the dual failure that does this unnecessarily a "Type 2" failure (a failure of commission).

Although we refer to Type 1 and Type 2 failures as duals, their treatment requires careful consideration because the demands and nondemands that underlie them are somewhat different in character. Intuitively, a demand is an event at some *point* in time, whereas a nondemand is the absence of a demand over a *period* of time. We unify the two by adjusting our treatment to consider the *rate* at which demands occur.

The channels of the systems we are interested in typically operate as cyclic processes executing at some fixed rate, such as so many times a second; on each execution, they sample sensors, perform some computation, possibly update their internal state, and decide whether to act (e.g., to shut down the reactor) or not. As time passes, there will be very many executions, only a few of which might be demands—i.e., represent states in which the channels are required to act. We suppose that whether any particular execution is a demand or not is independent of any other execution, so the numbers of demands in disjoint time intervals are independent of each other (i.e., "independent increments" in the terminology of statistics) and the probability distribution of the number of demands in a time interval depends only on the length of the interval (i.e., "stationary increments"). Under these conditions, the occurrence of demands will be approximately a Poisson process with rate

d, say. For the applications we have in mind, there will be very great disparity between the durations of the cycles and the expected times between successive demands, so this approximation will be very close. In what follows, therefore, we shall use the continuous time formalism of the Poisson process. So

$$d = \lim_{\delta t \to 0} \frac{P(1 \text{ demand in } (t,\, t+\delta t))}{\delta t}. \tag{4.1}$$

We define failure rates in a similar way. Then, for Type 1 failures we have

$$\begin{aligned}
&\text{1oo2 system Type 1 failure rate} \\
&= \lim_{\delta t \to 0} \frac{P(1 \text{ demand and system fails in } (t,\, t+\delta t))}{\delta t} \\
&= \lim_{\delta t \to 0} \frac{P(\text{system fails} \,|\, 1 \text{ demand in } (t,\, t+\delta t)) \times P(1 \text{ demand in } (t,\, t+\delta t))}{\delta t} \\
&= \lim_{\delta t \to 0} P(\text{system fails} \,|\, 1 \text{ demand in } (t,\, t+\delta t)) \times \frac{P(1 \text{ demand in } (t,\, t+\delta t))}{\delta t} \\
&\leq \left(C + (1-C) \times P_A^* \times P_B^*\right) \times d
\end{aligned}$$

where the first term comes from the epistemic formula (2.12) and the second from (4.1). Using (2.13) and rearranging terms we also have the conservative simplification

$$\text{1oo2 system Type 1 failure rate} \leq d \times (C + P_A \times P_B). \tag{4.2}$$

For Type 2 failures due to the A channel, we have

$$\begin{aligned}
&\text{1oo2 system Type 2 failure rate due to } A \\
&= \lim_{\delta t \to 0} \frac{P(\text{no demand and } A \text{ activates in } (t,\, t+\delta t))}{\delta t} \\
&= \lim_{\delta t \to 0} \frac{P(A \text{ activates} \,|\, \text{no demand in } (t,\, t+\delta t)) \times P(\text{no demand in } (t,\, t+\delta t))}{\delta t} \\
&= \lim_{\delta t \to 0} \frac{P(A \text{ activates} \,|\, \text{no demand in } (t,\, t+\delta t))}{\delta t} \times P(\text{no demand in } (t,\, t+\delta t)).
\end{aligned}$$

The second term here is conservatively approximated by 1. Then, taking limits, the first term is the failure rate of Channel A with respect to Type 2 failures, whose epistemic value (i.e., mean of the assessor's posterior distribution) we will denote by F_{A2}. Thus, we have

$$\text{1oo2 system Type 2 failure rate due to } A \leq F_{A2}. \tag{4.3}$$

We could perform an exactly similar calculation for Type 2 failures due to the B channel, but this would not exploit the possible perfection of that channel. Accordingly, we suppose that the possible perfection of B extends to Type 2 failures and that p_{B2} is its aleatory probability of imperfection with respect to those failures. We then have

$$\begin{aligned}
&P(B \text{ activates} \,|\, \text{no demand in } (t,\, t+\delta t)) \\
&= P(B \text{ activates} \,|\, B \text{ imperfect, no demand in } (t,\, t+\delta t)) \times P(B \text{ imperfect}) \\
&\quad + P(B \text{ activates} \,|\, B \text{ perfect, no demand in } (t,\, t+\delta t)) \times P(B \text{ perfect}).
\end{aligned}$$

The second term in this sum is zero (B does not activate on a nondemand if it is perfect with respect to Type 2 failures), so the right hand side reduces to

$$P(B \text{ activates} \,|\, B \text{ imperfect, no demand in } (t,\, t+\delta t)) \times p_{B2}.$$

The calculation given earlier for the A channel can now be reproduced for the B channel, but working at the aleatory level and substituting the expression above in the final line. This gives

$$\begin{aligned}&\text{1oo2 system Type 2 aleatory failure rate due to } B\\ &\leq \lim_{\delta t \to 0} \frac{P(B \text{ activates} \,|\, B \text{ imperfect, no demand in } (t,\, t+\delta t))}{\delta t} \times P(\text{no demand in } (t,\, t+\delta t)) \times p_{B2}.\end{aligned}$$

Here, the second term is conservatively approximated by 1. Then, taking limits, the first term is the aleatory failure rate of B with respect to Type 2 failures, *conditional* on B being imperfect with respect to these failures, which we will denote by $f_{B2|np}$.

$$\text{1oo2 system Type 2 aleatory failure rate due to } B \leq f_{B2|np} \times p_{B2}. \tag{4.4}$$

The right hand side is the product of two parameters describing aleatory uncertainty. We now have to consider the assessor's epistemic uncertainty about these parameters to derive a posterior expression for the failure rate. A fully accurate treatment requires integrating with respect to a joint probability function $G(f_{B2|np},\, p_{B2})$ just as we did with the function $F(p_A,\, p_B)$ in Section 2.2. However, it seems plausible that an assessor's beliefs about these parameters will be independent (why should the rate of failure, if imperfect, be dependent on the probability of imperfection?) so that the distribution G can be factorized and the failure rate due to Type 2 errors in B is then given by

$$\text{1oo2 system Type 2 failure rate due to } B \leq F_{B2|np} \times P_{B2} \tag{4.5}$$

where $F_{B2|np}$ is the assessor's posterior failure rate of B with respect to Type 2 failures, assuming B is imperfect with respect to these failures, and P_{B2} is the posterior probability of B being imperfect with respect to Type 2 failures. Each of these is the mean of the posterior distribution of its random variable considered alone.

We can now assess the overall merit of a 1oo2 system in terms of its *risk* per unit time, taking account of both Type 1 and Type 2 failures, as follows. Denote the consequence (cost) of a Type 1 failure by c_1, and the consequences of Type 2 failures by the A and B channels by c_{A2} and c_{B2}, respectively.[1] In some cases, a Type 2 failure may have no safety consequences, so c_{A2} and c_{B2} can be assessed as zero. But even in these cases there are likely to be economic losses associated with a Type 2 failure and these could be included in a broader calculation of risk, resulting in nonzero assessments for c_{A2} and c_{B2}.

The rates for the three kinds of system failure are given by (4.2), (4.3), and (4.5), respectively. Adding the products of these and their costs yields the total risk:

$$\text{1oo2 } \textit{risk} \text{ per unit time} \leq c_1 \times d \times (C + P_{A1} \times P_{B1}) + c_{A2} \times F_{A2} + c_{B2} \times F_{B2|np} \times P_{B2}. \tag{4.6}$$

[1] Since the A and B channels may bring the system to a safe state in different ways (e.g., by dropping the control rods or by poisoning the moderator, respectively), it is reasonable to assess different costs or severity of consequences to their Type 2 failures.

We may assume that the cost c_1 of a Type 1 failure is large, and hence that extensive design and engineering effort is focused on ensuring that both the demand rate d and the failure rate $(C + P_{A1} \times P_{B1})$ are small, so that the contribution to overall risk by the first term in this sum is acceptable.

The second and third terms in the sum (4.6) are not reduced by a demand rate so, when c_{A2} and c_{B2} are nonzero, their contribution to overall risk will be small only if all their other components are small. The risk from a Type 2 failure of the main A channel is inherent in any safety system, so we may assume that the failure rate F_{A2} is acceptable, relative to the cost of this type of failure; the risk due to a Type 2 failure of the B channel, however, is an additional factor introduced by the decision to employ a 1oo2 architecture. Hence, we will need to be sure that the contribution from the product $c_{B2} \times F_{B2|np} \times P_{B2}$ is an acceptable price to pay for the reduction in risk that the architecture brings to Type 1 failures.

In the above analysis we have assumed that d, c_1, c_{A2} and c_{B2} are known. If they are not, then they need to be treated as a part of a more general epistemic analysis. This is simple if it can be assumed, as may be reasonable in some cases, that the assessor's beliefs about these parameters are independent of his beliefs about the parameters representing different probabilities and rates of failure and imperfection. We shall not pursue this further here.

Chapter 5

Monitored Architectures

In this chapter we examine a different class of architectures from the 1oo2 case considered so far. These architectures employ an *operational* channel that is completely responsible for the functions of the system concerned, and a *monitor* channel that can signal an "alarm" if it deems the operational channel to have failed or to be causing unsafe behavior. Other systems will ignore the outputs of a system whose monitor is signaling an alarm, so the monitor transforms potential "malfunction" or "unintended function" into "loss of function," which is generally a less serious failure condition. Loss of function may still be a dangerous condition, so there must generally be some larger system that responds to the alarm signal and compensates in some way for the lost function. This is really, therefore, an architecture for *sub*systems; we refer to it as a *monitored architecture*.

Monitored architectures are attractive for subsystems of highly redundant systems where occasional loss of function is anticipated and it is likely that some other subsystem can take over the functions of the one whose monitor has signaled the alarm, and the latter subsystem can be shut down or otherwise isolated.

Aircraft systems have this characteristic: the redundancy (e.g., multiple engines, multiple sources of air data, and internal redundancy in systems such as autopilot, flight management, autoland, autothrottle, FADEC, etc.) is primarily present to cope with physical failures (of a mechanical element or of a computer),[1] but there is increasing interest in using it to handle software faults. The software in flight-critical functions is supposed to be almost certainly fault-free and guidelines such as ARP4754 [54], DO-178B [43], and DO-297 [44] are intended to ensure that it is so. However, the reliability requirements for flight critical functions (those that could cause catastrophic failure conditions) are so high (e.g., $1 - 10^{-9}$ per hour, sustained for the duration of the flight [15, paragraph 10.b])[2] that there is some concern that no amount of development assurance can deliver sufficient confidence that its embedded software is free of faults.

[1] There is a requirement that no single failure can cause a "catastrophic failure condition."

[2] An explanation for this figure can be derived [29, page 37] by considering a fleet of 100 aircraft, each flying 3,000 hours per year over a lifetime of 33 years (thereby accumulating about 10^7 flight-hours). If hazard analysis reveals ten potentially catastrophic failure conditions in each of ten systems, then the "budget" for each is about 10^{-9} if such a condition is not expected to occur in the lifetime of the fleet.

A position paper by a team of experts representing certification authorities of North and South America, and Europe [9] expresses this concern,[3] and suggests that "fault-tolerance techniques should be applied (such as, diverse and redundant implementations or simple design)..." Monitored architectures are an approach for achieving this and our proposal is that the monitor channel should be supported by a credible claim of possible perfection. One way to do this would be to develop the monitor channel as a very simple piece of software and to formally verify it against its requirements, in much the same way as we described for the possibly perfect channel in the 1oo2 architecture.

Alternatively, rather than formally analyzing the monitor channel, we could formally *synthesize* it from its requirements. There is a topic in formal methods known as *Runtime Verification*, whose technology provides methods for automated synthesis of monitors for formally specified properties [20]. Given requirements specified in a language such as EAGLE or RULER [4], the methods of runtime verification can automatically generate an efficient (and provably correct) monitor that will raise an alarm as soon as a requirement is violated.

We refer to monitors developed using either formal analysis or synthesis as *formal monitors*, and now turn to consideration of the requirements that they monitor.

One obvious source of requirements is the documentation for the operational channel being monitored. The problem with this choice is that these requirements are generally at the unit level and the operational software is often robustly correct at this level (due to the effectiveness of practices such as those recommended by DO-178B). That is to say, the (sub)system may have failed or be operating in an unsafe manner, but each individual software unit is operating correctly according to its unit-level requirements.

A recent in-flight incident illustrates this topic. It concerns an Airbus A340-642, registration G-VATL, which suffered a fuel emergency on 8 February 2005 [55]. The plane was over Europe on a flight from Hong Kong to London when two engines flamed out. The crew found that the fuel tanks supplying those engines were empty and those for the other two engines were very low. They declared an emergency and landed at Amsterdam. The subsequent investigation reported that two Fuel Control Monitoring Computers (FCMCs) are responsible for pumping fuel between the tanks on this type of airplane. The two FCMCs cross-compare and the "healthiest" one drives the outputs to the data bus. In this case, both FCMCs had known faults (but complied with the minimum capabilities required for flight); unfortunately, one of the faults in the one judged healthiest was the inability to drive the data bus. Thus, although it gave correct commands to the fuel pumps (there was plenty of fuel distributed in other tanks), these were never received. A low fuel warning should have been generated by a backup system, but this warning is disregarded by the display system unless both FCMCs have failed.[4] Monitoring low-level requirements for the FCMCs would not detect this problem, since faulty requirements were the root of the problem (i.e., they precipitated a common-cause failure).

[3] Notice that this can be interpreted as "claim limit," though of a different kind from that used in nuclear systems.

[4] The reason for this is not explained, but a plausible explanation is that it is to avoid Type 2 failures. One of the recommendations of the Incident Report [55, Safety Recommendation 2005-37] is to change this so that the backup can always trigger a low fuel level warning.

This example illustrates that there is unlikely to be much benefit in monitoring requirements at or below the unit level: not only is critical software generally correct with respect to this level of specification, but larger problems may not be manifested at this level (i.e., they are *emergent* to the unit level). Instead, we need to monitor properties that more directly relate to the safe functioning of the system, and that are more likely to be violated when problems are present.

The claims and assumptions of an argument-based assurance case can provide exactly these properties. An assurance case for the FCMCs would surely include statements about the acceptable distribution of fuel among the different tanks, and minimum levels in the tanks feeding the engines. These properties are different in kind (i.e., "diverse") from those that would appear in software requirements, which typically focus on tasks to be performed, rather than properties to be maintained.

A monitored architecture therefore has a highly reliable operational channel, whose behavior is derived from its software requirements specification, and a simple formal monitor that has a high probability of perfection with respect to properties from the system assurance case.[5] Analysis of the reliability of such architectures can build on our previous analysis of 1oo2 architectures, with the operational channel taking the part of A and the monitor the part of B, but with some changes that reflect differences in the architectures.

One significant change is that there is no notion of "demand" for the operational channel in the monitored architecture; instead, it continuously performs its assigned operational role, but may occasionally suffer failures that violate a safety requirement. Thus, a monitored architecture has a Type 1 failure if its operational channel violates a safety requirement and its monitor fails to detect that fact. The operational channel will have some failure rate, denoted fr_A, the monitor will have some probability of imperfection with respect to Type 1 failures (i.e., failures of omission), denoted pnp_{B1}, and we seek a rate for Type 1 system failure based on these parameters. The other significant change is that there is no notion of Type 2 failure (i.e., of commission) for the operational channel: only the monitor channel can raise an alarm, and thereby cause a Type 2 system failure by doing so unnecessarily. We suppose that the monitor will have some probability of imperfection with respect to Type 2 failures, denoted pnp_{B2} and we seek a rate for Type 2 system failure based on this parameter.

[5] Of course, the assurance case should provide evidence and argument that the operational channel does satisfy those properties. Thus, the monitor is redundant if the assurance case is sound; its justification is diversity.

We begin with the aleatory analysis for Type 1 system failure; we assume that fr_A has known value f_A and that pnp_{B1} has known value p_{B1}. Then

$$\begin{aligned}
&P(\text{monitored architecture has Type 1 failure in } (t,\, t+\delta t) \,|\, fr_A = f_A,\, pnp_{B1} = p_{B1}) \\
&\quad = \; P(A \text{ fails in } (t,\, t+\delta t), B \text{ does not detect failure} \,|\, fr_A = f_A,\, pnp_{B1} = p_{B1}) \\
&\quad \leq \; P(A \text{ fails in } (t,\, t+\delta t), B \text{ imperfect} \,|\, fr_A = f_A,\, pnp_{B1} = p_{B1}) \qquad (5.1) \\
&\quad = \; P(A \text{ fails in } (t,\, t+\delta t) \,|\, B \text{ imperfect}, fr_A = f_A,\, pnp_{B1} = p_{B1}) \\
&\qquad \times P(B \text{ imperfect} \,|\, fr_A = f_A,\, pnp_{B1} = p_{B1}) \\
&\quad = \; P(A \text{ fails in } (t,\, t+\delta t) \,|\, fr_A = f_A,\, pnp_{B1} = p_{B1}) \qquad (5.2) \\
&\qquad \times P(B \text{ imperfect} \,|\, fr_A = f_A,\, pnp_{B1} = p_{B1}) \\
&\quad = \; P(A \text{ fails in } (t,\, t+\delta t) \,|\, fr_A = f_A,\, pnp_{B1} = p_{B1}) \times p_{B1}. \qquad (5.3)
\end{aligned}$$

Line (5.2) follows from the one above it because the fact that B is imperfect tells us nothing about the failure of A in this interval. Line (5.1) follows from the one above it by the following reasoning:

$$\begin{aligned}
&P(A \text{ fails in } (t,\, t+\delta t), B \text{ imperfect} \,|\, fr_A = f_A,\, pnp_{B1} = p_{B1}) \\
&\quad = \; P(A \text{ fails in } (t,\, t+\delta t), B \text{ imperfect and does not detect failure} \,|\, fr_A = f_A,\, pnp_{B1} = p_{B1}) \\
&\qquad + P(A \text{ fails in } (t,\, t+\delta t), B \text{ imperfect and detects failure} \,|\, fr_A = f_A,\, pnp_{B1} = p_{B1}) \\
&\quad > \; P(A \text{ fails in } (t,\, t+\delta t), B \text{ imperfect and does not detect failure} \,|\, fr_A = f_A,\, pnp_{B1} = p_{B1}) \\
&\quad = \; P(A \text{ fails in } (t,\, t+\delta t), B \text{ does not detect failure} \,|\, fr_A = f_A,\, pnp_{B1} = p_{B1}).
\end{aligned}$$

The last line above follows from its predecessor because B must be imperfect if it does not detect failure (of A).

From (5.3), dividing by δt and taking limits, we obtain

$$\text{Monitored architecture Type 1 aleatory failure rate} \leq f_A \times p_{B1}. \qquad (5.4)$$

The Type 2 failure rate for the monitored architecture is just that for the monitor channel. We can directly apply the analysis from the 1oo2 case so that (4.4) becomes

$$\text{Monitored architecture Type 2 aleatory failure rate} \leq f_{B2|np} \times p_{B2} \qquad (5.5)$$

where $f_{B2|np}$ is the Type 2 failure rate for the monitor channel, conditional on its imperfection with respect to that class of failures.

As with the cases examined earlier, we now need to consider epistemic uncertainty in the two formulas above. For the Type 1 failure rate (5.4), we will need to consider a joint distribution function $H(f_A,\, p_{B1})$ representing the assessor's beliefs about these variables. The first of these is the failure rate of the operational channel, while the second is the probability of imperfection of the monitor. We argue that these are different measures about very different channels: one a fairly complex system designed to meet its requirements, the other a very simple check of properties taken directly from the safety case. Thus, it is very plausible that the assessor's beliefs about these channels will be independent, and so the function $H(f_A,\, p_{B1})$ can be factorized into distributions for the individual variables.

However, although the channels are diverse in purpose and design, it is quite likely they will share some implementation mechanisms. For example, a monitor for the A340 fuel system might rely on the same sensors as those used by the operational channel.

One approach would be to explicitly factor the system into three components—shared mechanism (e.g., sensors), operational channel, and monitor—and to calculate probabilities over this more complex model. However, we propose a simpler treatment that is similar to that employed to yield (2.13) in Section 2.2. There, we introduced an element C to absorb failures due to faults in requirements and interpretation that are common to both channels; here, we introduce an element M_1 to absorb failures due to mechanisms that are common to both channels. Thus,

$$\text{Monitored architecture Type 1 failure rate} \leq M_1 + F_A \times P_{B1}, \tag{5.6}$$

where M_1 is the assessed Type 1 failure rate due to mechanisms shared between the operational and monitor channels, F_A is the posterior failure rate of the operational channel (excluding items in M_1), and P_{B1} is the posterior probability of imperfection (with respect to Type 1 failures) of the monitor channel (again, excluding items in M_1).

The Type 2 failure rate (5.5) was derived directly from the B channel case in a 1oo2 architecture (4.4). We argue that the epistemic valuation can similarly be derived from (4.5) in the 1oo2 case to yield

$$\text{Monitored architecture Type 2 failure rate} \leq M_2 + F_{B2|np} \times P_{B2}, \tag{5.7}$$

where M_2 is the assessed Type 2 failure rate due to mechanisms shared between the operational and monitor channels, $F_{B2|np}$ is the posterior Type 2 failure rate for the monitor channel, assuming it is imperfect with respect to that class of failures, and P_{B2} is the posterior probability of imperfection of the monitor channel with respect to Type 2 failures. (Although Type 2 failures are due to the monitor channel alone, we must still include the contributions of the mechanisms that it shares with the operational channel.)

Putting these together, we obtain the *risk* of a monitored architecture:

$$\text{Monitored architecture } \textit{risk} \text{ per unit time} \leq c_1 \times (M_1 + F_A \times P_{B1}) + c_2 \times (M_2 + F_{B2|np} \times P_{B2}), \tag{5.8}$$

where c_1 is the cost of a Type 1 failure and c_2 the cost of a Type 2 failure.

Assuming, as seems likely, that M_1 is much smaller than F_A, a relatively modest claim for perfection of the monitor with respect to Type 1 failures (e.g., $P_{B1} = 10^{-3}$) provides significant reduction in the overall risk due to Type 1 failures.

Employing a monitor brings with it the new risk of a Type 2 failure, and responsibility for this is borne by the monitor alone. For this new risk to be acceptable, assuming M_2 is small, we need either a strong claim for perfection of the monitor with respect to Type 2 failures (e.g., $P_{B2} = 10^{-6}$), or the consequence of a Type 2 failure must be modest. Some relief from these demanding requirements is possible if the assessor is able to assign a small value to $F_{B2|np}$, the failure rate of the monitor channel on the assumption that it is imperfect. Statistically valid testing of the monitor channel provides a basis for assessing

$F_{B2|np}$ and could yield values on the order of 10^{-3}. Obviously, the tests must reveal no failures, since then the channel would definitely be imperfect.

Whatever the contribution of $F_{B2|np}$, there should be an inverse relationship between the product $F_{B2|np} \times P_{B2}$ and the cost c_2. For the case of the A340 FCMC failure, it is likely that the response to an alarm raised by a monitor would be a warning to the pilots, who would then follow the standard troubleshooting and emergency procedures established for possible faults in the fuel system. Thus, the consequences of a false alarm are modest and commensurate with a correspondingly modest claim for perfection of the monitor.

On the other hand, monitoring mechanisms on American Airlines Flight 903 of 12 May 1997 (an Airbus A300) inappropriately raised an alarm indicating failure of one of the avionics buses. This caused an automated bus reset, which in turn caused the Electronic Flight Instrumentation System (EFIS, which is responsible for displaying instruments on the cockpit monitors) to go blank for a while. At that time, the pilots were attempting to recover from a major upset (the plane was in a succession of stalls) and the loss of all primary instruments at this critical time jeopardized the recovery [32]. Here, the cost of a Type 2 error was very high and it would seem that an extremely low probability of imperfection should have been required of the monitor.

In fact, the monitor did not fail (which is why we called its action "inappropriate" rather than "incorrect"): the problem was in the properties that it was required to monitor. Until now, we have implicitly considered demands to be events in the physical world—but such events do not always announce themselves unambiguously: their detection may require delicate interpretation of sensors or indirect inference from other available data. In this case, the physical event to be monitored is failure of an avionics bus, and one way to detect this event is reception of faulty data delivered by the bus. And one way to judge whether data is faulty is by considering its physical plausibility. Here, roll angle rates of change greater than 40 degrees/second were considered implausible, and reception of sensor values in excess of this limit were what triggered the monitor. But, as noted, the airplane was stalled, the roll rates were real, and the inference of a bus fault was incorrect.

Designers choose the requirements for properties to be monitored; these properties, not the physical events they are intended to detect, are the demands that a monitor responds to and for which its probability of perfection is assessed. Through consideration of the possible costs of appropriate and inappropriate actions and inactions, designers may choose to fine-tune detection thresholds and adjust other elements in the specifications of properties to be monitored. They can also adjust the actions taken in response to violation of these properties. The consequences of these choices can far outweigh the perfection or otherwise of the monitor—hence, there is little point in requiring high probability of perfection in a monitor, when the properties that it monitors are imperfect indicators of the physical event that it should respond to. Notice that this concern is not unique to monitors: it applies just the same to 1oo2 architectures.

It might seem reasonable to extend the probability of perfection for a monitor to include "perfection" of the monitored properties (relative the physical events they are intended to identify). The aleatory analysis would be unchanged, but now provides a more comprehensive and holistic bound on overall system reliability and risk. The difficulty arises at the epistemic stage, for there is no credible way to estimate a probability of perfection for

the properties being monitored. In fact, perfection is an inappropriate notion to apply to these properties: they are internal models of an external reality and the appropriate notion is one of "fidelity," which is measured as reliability. Observe that the failure rate of the full system is then approximately the sum of the failure rates (i.e., unreliabilities) of the implementation of the monitor and of the properties that it monitors.

For this reason, monitors and their required probabilities of perfection, the reliability of the properties that they monitor, and the costs of the responses that they trigger and of the failures that they avert, should be developed in the context of a safety case, where the necessary tradeoffs can be made in an integrated manner. In the case of Flight 903, the specification for a demand should probably have been made more stringent, and the response should probably have been less Draconian. Then, a modest and achievable probability of perfection for the monitor could have delivered a useful and credible claim to the safety case.

Chapter 6

Conclusion

We have considered systems in which a highly reliable software component or "channel" A operates in combination with a channel B that supports a credible claim of "perfection," meaning that it will never fail. The claim of perfection may be wrong, so we speak of a "possibly perfect" channel characterized by a probability of imperfection. Our results for this model clarify, generalize, and extend an earlier result in [25].

The clarification lies in showing that there is *conditional independence* between the events "channel A fails on a randomly selected demand" and "channel B is imperfect." This means that the description of the aleatoric uncertainty is particularly simple: under conservative assumptions, the conditional system *pfd* is bounded by $p_A \times p_B$, where p_A is the probability that A fails on a randomly selected demand, and p_B is the probability that B is imperfect.

This contrasts with the usual approach involving channels for which claims of perfection cannot be made and only reliability is claimed; here, independence of channel failures *cannot* be asserted, and thus the two channel *pfd*s are not sufficient to characterize the model. Instead, consideration needs to be given to the "degree" of dependence of the failures of the two channels, and this is extremely difficult to incorporate into the model: in the Eckhardt and Lee [14] and Littlewood and Miller [24] models, for example, it involves knowing the covariance of the "difficulty functions" over the demand space. Typically, these are *not* known, and cannot easily be estimated.

The generalization lies in the treatment of epistemic uncertainty, which is shown to be the *sole* source of dependence in this model. An assessor has to describe his beliefs about the two parameters of the model via the "belief" distribution $F(p_A, p_B)$ in order to obtain his unconditional probability of failure of the system on a randomly selected demand. Representing beliefs as a bivariate distribution may not be easy, but supposing that the distribution can be factored will usually not accurately represent the assessor's beliefs. We therefore proposed a conservative approach that is similar to that underpinning some "claim limit" approaches in assurance cases, and that can be seen as one way of formalizing such arguments. It requires the assessor to represent his beliefs in terms of just three quantities: the probability C of common faults that afflict both channels, the probability P_A^* of failure for the reliable channel A in the absence of such common faults,

and the probability P_B^* of imperfection for the possibly perfect channel, again assuming the absence of common faults. The probability that the system fails on a randomly selected demand is then conservatively bounded by

$$C + (1 - C) \times P_A^* \times P_B^*.$$

The model described here seems much simpler than the classical one (with two reliable channels) in the treatment of both aleatory and epistemic uncertainty. We described the context of argument-based assurance cases in which the epistemic probabilities C, P_A^*, and P_B^* must be formulated, and argued that credible and useful values can be constructed with modest enhancements to current development and assurance practices. Development of a channel that may claim perfection is most plausible when the channel is very simple, or is subjected to mechanically checked formal verification or, preferably, both. We considered fallibilities in formal verification and argued that these can be accommodated within useful and credible values for P_B^*.

The earlier result has been extended by considering failures of commission (e.g., tripping a reactor when this is not needed), and then further extended by combining these results with a modification of the 1oo2 analysis to yield an analysis for monitored architectures: those, such as are proposed for aircraft, in which a reliable operational channel is monitored by a possibly perfect channel that can trigger higher-level fault recovery. These cases require extension of the statistical model from one based on probability of failure on demand to one based on rates of failure.

Like our first result, these extended results are relatively simple and require estimation of only a few parameters. The simplicity is achieved through conservative assumptions; hence our results are conservative also, possibly very much so. Nonetheless, plausible values for the parameters yield attractive conclusions.

The thrust of this report has been about *assessment* of reliability of fault tolerant diverse software-based systems. The problem of assessing the reliability of such a system is often more difficult than the problem of *achieving* that reliability, particularly when the reliability requirements are very stringent. It is worth saying, then, that the asymmetric 1oo2 architecture considered here is one that has long been regarded as an attractive one for achieving reliability in certain applications (e.g., nuclear reactor protection systems), not least because it forces the reduction of excessive functionality, with its ensuing complexity, which are known enemies of dependability. That this architecture brings greater simplicity to the assessment process is an added bonus.

1oo2 architectures are most appropriate for systems that have a safe state; monitored architectures are one attempt to extend these benefits to systems that must continue operation in the presence of faults.

Monitored architectures are recommended in the relevant aircraft guidelines [54, Table 5–2] as one way to achieve the high levels of dependability required for flight critical software. For example, it is suggested that a Level A system (one requiring the highest level of assurance) can be achieved by a Level C operational channel and a Level A monitor. Current revisions to these guidelines are augmenting its Table 5–2 with rules that allow calculation of acceptable alternatives. Our results can be seen as a way to provide formal underpinnings to

those rules. We showed that risk due to failure of the operational channel can be significantly reduced by a monitor, but that the probability of imperfection in the monitor with respect to faults of commission must be carefully weighed against the danger and cost of unnecessary fault recovery. Furthermore, a perfect monitor is only as good as the properties that it monitors, so a large part of the challenge in developing effective monitors is to ensure that the properties chosen for monitoring are good ones.

For the future, it will be interesting to extend the analysis of monitored architectures to include recovery mechanisms, and possibly to consider a basis for certification of systems that employ adaptive control or other techniques that are currently beyond the pale.

Acknowledgements

We would like to thank the following colleagues for valuable discussions about earlier versions of this work: Peter Bishop, Robin Bloomfield, Alan Burns, Bruno Dutertre, Pat Lincoln, Paul Miner, Peter Popov, Andrey Povyakalo, Bob Riemenschneider, N. Shankar, Lorenzo Strigini, David Wright, and Bob Yates.

The first author's work was supported by the INDEED project, funded by the UK Engineering and Physical Sciences Research Council; and by the DISPO project, funded by British Energy, NDA (Sellafield, Magnox North, Magnox South), AWE and Westinghouse under the CINIF Nuclear Research Programme (the views expressed in this Report are those of the authors and do not necessarily represent the views of the members of the Parties; the Parties do not accept liability for any damage or loss incurred as a result of the information contained in this Report).

The second author was supported by NASA cooperative agreements NNX08AC64A and NNX08AY53A, and by National Science Foundation grant CNS-0720908.

Bibliography

[1] *In-Flight Upset Event, 240 km North-West of Perth, WA, Boeing Company 777-200, 9M-MRG, 1 August 2005.* Australian Transport Safety Bureau, March 2007. Reference number Mar2007/DOTARS 50165, available at `http://www.atsb.gov.au/publications/investigation_reports/2005/AAIR/aair200503722.aspx`.

[2] *In-Flight Upset Event, 154 km West of Learmonth, WA, 7 October 2008, VH-QPA Airbus A330-303.* Australian Transport Safety Bureau, March 2009. Reference number AO-2008-070 Interim Factual, available at `http://www.atsb.gov.au/publications/investigation_reports/2008/AAIR/pdf/AO2008070_interim.pdf`.

[3] *Licensing of Safety Critical Software for Nuclear Reactors: Common Position of Seven European Nuclear Regulators and Authorised Technical Support Organizations.* AVN Belgium, BfS Germany, CSN Spain, ISTec Germany, NII United Kingdom, SKI Sweden, STUK Finland, 2007. Available at `http://www.bfs.de/de/kerntechnik/sicherheit/Licensing_safety_critical_software.pdf`.

[4] Howard Barringer, David Rydeheard, and Klaus Havelund. Rule systems for run-time monitoring: From EAGLE to RULER. In *Runtime Verification (RV 2007)*, Volume 4839 of Springer-Verlag *Lecture Notes in Computer Science*, pages 111–125, Vancouver, British Columbia, Canada, March 2007.

[5] Günther Bauer, Hermann Kopetz, and Wilfried Steiner. The central guardian approach to enforce fault isolation in the time-triggered architecture. In *The Sixth International Symposium on Autonomous Decentralized Systems*, pages 37–44, IEEE Computer Society, Pisa, Italy, April 2003.

[6] Robin Bloomfield and Bev Littlewood. Multi-legged arguments: The impact of diversity upon confidence in dependability arguments. In *The International Conference on Dependable Systems and Networks*, pages 25–34, IEEE Computer Society, San Francisco, CA, June 2003.

[7] *Statistical Summary of Commercial Jet Airplane Accidents, Worldwide Operations 1959-2007.* Boeing Commercial Airplane Group, Seattle, WA, July 2008. Available at `http://www.boeing.com/news/techissues/pdf/statsum.pdf`.

[8] Ricky W. Butler and George B. Finelli. The infeasibility of experimental quantification of life-critical software reliability. *IEEE Transactions on Software Engineering*, 19(1):3–12, January 1993.

[9] *CAST Position Paper 24: Reliance on Development Assurance Alone when Performing a Complex and Full-Time Critical Function*. Certification Authorities Software Team (CAST), March 2006. Available from `http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/cast_papers/`.

[10] C. Cornes, J. Courant, J.C. Filliâtre, G. Huet, P. Manoury, C. Paulin-Mohring, C. Muñoz, C. Murthy, C. Parent, A. Saibi, and B. Werner. The Coq proof assistant reference manual, version 5.10. Technical report, INRIA, Rocquencourt, France, February 1995. Coq home page: `http:///coq.inria.fr`.

[11] Judy Crow, Sam Owre, John Rushby, N. Shankar, and Dave Stringer-Calvert. Evaluating, testing, and animating PVS specifications. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, March 2001. Available from `http://www.csl.sri.com/users/rushby/abstracts/attachments`.

[12] Kevin Driscoll, Brendan Hall, Håkan Sivencrona, and Phil Zumsteg. Byzantine fault tolerance, from theory to reality. In Stuart Anderson, Massimo Felici, and Bev Littlewood, editors, SafeComp *2003: Proceedings of the 22nd International Conference on Computer Safety, Reliability, and Security*, pages 235–248, Edinburgh, Scotland, September 2003.

[13] Dave E. Eckhardt, Alper K. Caglayan, John C. Knight, Larry D. Lee, David F. McAllister, Mladen A. Vouk, and John P. J. Kelly. An experimental evaluation of software redundancy as a strategy for improving reliability. *IEEE Transactions on Software Engineering*, 17(7):692–702, July 1991.

[14] Dave E. Eckhardt, Jr. and Larry D. Lee. A theoretical basis for the analysis of multiversion software subject to coincident errors. *IEEE Transactions on Software Engineering*, SE-11(12):1511–1517, December 1985.

[15] *System Design and Analysis*. Federal Aviation Administration, June 21, 1988. Advisory Circular 25.1309-1A.

[16] M. Gordon, R. Milner, and C. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.

[17] M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic*. Cambridge University Press, Cambridge, UK, 1993. HOL home page: `http://www.cl.cam.ac.uk/Research/HVG/HOL/`.

[18] Grégoire Hamon, Leonardo de Moura, and John Rushby. Automated test generation with SAL. Technical note, Computer Science Laboratory, SRI International, Menlo Park, CA, September 2005. Available at `http://www.csl.sri.com/users/rushby/abstracts/sal-atg`.

[19] John Harrison. Towards self-verification of HOL Light. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*. Volume 4130 of Springer *Lecture Notes in Computer Science*, pages 177–191, 2006.

[20] Klaus Havelund and Grigore Rosu. Efficient monitoring of safety properties. *Software Tools for Technology Transfer*, 6(2):158–173, August 2004.

[21] Matt Kaufmann and J Strother Moore. An industrial strength theorem prover for a logic based on Common Lisp. *IEEE Transactions on Software Engineering*, 23(4):203–213, April 1997. ACL2 home page: `http://www.cs.utexas.edu/users/moore/acl2/`.

[22] J. C. Knight and N. G. Leveson. An empirical study of failure probabilities in multi-version software. In *Fault Tolerant Computing Symposium 16*, pages 165–170, IEEE Computer Society, Vienna, Austria, July 1986.

[23] J. C. Knight and N. G. Leveson. An experimental evaluation of the assumption of independence in multiversion programming. *IEEE Transactions on Software Engineering*, SE-12(1):96–109, January 1986.

[24] B. Littlewood and D. R. Miller. Conceptual modeling of coincident failures in multiversion software. *IEEE Transactions on Software Engineering*, 15(12):1596–1614, December 1989.

[25] Bev Littlewood. The use of proof in diversity arguments. *IEEE Transactions on Software Engineering*, 26(10):1022–1023, October 2000.

[26] Bev Littlewood, Peter T. Popov, Lorenzo Strigini, and Nick Shryane. Modelling the effects of combining diverse software fault detection techniques. *IEEE Transactions on Software Engineering*, 26(12):1157–1167, December 2000.

[27] Bev Littlewood and Lorenzo Strigini. Validation of ultrahigh dependability for software-based systems. *Communications of the ACM*, pages 69–80, November 1993.

[28] Bev Littlewood and David Wright. The use of multi-legged arguments to increase confidence in safety claims for software-based systems: a study based on a BBN analysis of an idealised example. *IEEE Transactions on Software Engineering*, 33(5):347–365, May 2007.

[29] E. Lloyd and W. Tye. *Systematic Safety: Safety Assessment of Aircraft Systems*. Civil Aviation Authority, London, England, 1982. Reprinted 1992.

[30] J. May, G. Hughes, and A. D. Lunn. Reliability estimation from appropriate testing of plant protection software. *IEE/BCS Software Engineering Journal*, 10(6):206–218, November 1995.

[31] J Strother Moore. A grand challenge proposal for formal methods: A verified stack. In *Formal Methods at the Crossroads: From Panacea to Foundational Support*, Volume

2757 of Springer-Verlag *Lecture Notes in Computer Science*, pages 161–172, Lisbon, Portugal, 2003. 10th Anniversary Colloquium of UNU/IIST the International Institute for Software Technology of The United Nations University.

[32] *Safety Recommendations A-98-3 through -5.* National Transportation Safety Board, Washington, DC, January 1998. Available at `http://www.ntsb.gov/Recs/letters/1998/A98_3_5.pdf`.

[33] *Safety Recommendation A-07-70 through -86.* National Transportation Safety Board, Washington, DC, October 2007. Available at `http://www.ntsb.gov/Recs/letters/2007/A07_70_86.pdf`.

[34] *Safety Recommendations A-07-65 through -69.* National Transportation Safety Board, Washington, DC, October 2007. Available at `http://www.ntsb.gov/recs/letters/2007/A07_65_69.pdf`.

[35] William L. Oberkampf and Jon C. Helton. Alternative representations of epistemic uncertainty. *Reliability Engineering and System Safety*, 85(1–3):1–10, 2004.

[36] Anthony O'Hagan, Caitlin E. Buck, Alireza Daneshkhah, J. Richard Eiser, Paul H. Garthwaite, David J. Jenkinson, Jeremy E. Oakley, and Tim Rakow. *Uncertain Judgements: Eliciting Experts' Probabilities.* Wiley, 2006.

[37] Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995. PVS home page: `http://pvs.csl.sri.com`.

[38] Sam Owre and N. Shankar. Theory interpretations in PVS. Technical Report SRI-CSL-01-01, Computer Science Laboratory, SRI International, Menlo Park, CA, April 2001.

[39] L. C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994. Isabelle home page: `http://www.cl.cam.ac.uk/research/hvg/Isabelle/`.

[40] Lee Pike. A note on inconsistent axioms in Rushby's "Systematic formal verification for fault-tolerant time-triggered algorithms". *IEEE Transactions on Software Engineering*, 32(5):347–348, May 2006.

[41] *Engineering Safety Management (The Yellow Book), Volumes 1 and 2, Fundamentals and Guidance, Issue 4.* Rail Safety and Standards Board, London, UK, 2007. Available from `http://www.yellowbook-rail.org.uk/site/the_yellow_book/the_yellow_book.html`.

[42] Sanjai Rayadurgam and Mats Heimdahl. Coverage based test-case generation using model checkers. In *Eighth International Conference and Workshop on the Engineering of Computer Based Systems (ECBS)*, pages 83–91, IEEE Computer Society, Washington DC, April 2001.

[43] *DO-178B: Software Considerations in Airborne Systems and Equipment Certification*. Requirements and Technical Concepts for Aviation, Washington, DC, December 1992. This document is known as EUROCAE ED-12B in Europe.

[44] *DO-297: Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations*. Requirements and Technical Concepts for Aviation, Washington, DC, November 2005. Also issued as EUROCAE ED-124 (2007).

[45] J. C. Rouquet and P. J. Traverse. Safe and reliable computing on board the Airbus and ATR aircraft. In *Safety of Computer Control Systems (SAFECOMP '86)*, Published by Pergamon for the International Federation of Automatic Control, IFAC, Sarlat, France, July 1986.

[46] John Rushby. Using model checking to help discover mode confusions and other automation surprises. *Reliability Engineering and System Safety*, 75(2):167–177, February 2002. Available at `http://www.csl.sri.com/users/rushby/abstracts/ress02`.

[47] John Rushby. Harnessing disruptive innovation in formal verification. In Dang Van Hung and Paritosh Pandya, editors, *Fourth International Conference on Software Engineering and Formal Methods (SEFM)*, pages 21–28, IEEE Computer Society, Pune, India, September 2006.

[48] John Rushby. Automated formal methods enter the mainstream. *Communications of the Computer Society of India*, 31(2):28–32, May 2007. Special Theme Issue on Formal Methods edited by Richard Banach. Archived in Journal of Universal Computer Science Vol. 13, No. 5, pp. 650–660, available at `http://www.jucs.org/jucs_13_5`.

[49] John Rushby. A safety-case approach for certifying adaptive systems. In *AIAA Infotech@Aerospace Conference*, American Institute of Aeronautics and Astronautics, Seattle, WA, April 2009. AIAA paper 2009-1992; available at `http://www.csl.sri.com/users/rushby/abstracts/aiaa09`.

[50] Mark Saaltink. Domain checking Z specifications. In C. Michael Holloway and Kelly J. Hayhurst, editors, *LFM' 97: Fourth NASA Langley Formal Methods Workshop*, pages 185–192, NASA Langley Research Center, Hampton, VA, September 1997. Available at `http://atb-www.larc.nasa.gov/Lfm97/proceedings/`.

[51] *Air Traffic Services Safety Requirements, CAP 670*. Safety Regulation Group, UK Civil Aviation Authority, June 2008. See Part B, Section 3, Systems Engineering SW01: Regulatory Objectives for Software Safety Assurance in ATS Equipment; Available at `http://www.caa.co.uk/docs/33/cap670.pdf`.

[52] Natarajan Shankar. Trust and automation in verification tools. In Sungdeok (Steve) Cha, Jin-Young Choi, Moonzoo Kim, Insup Lee, and Mahesh Viswanathan, editors, *6th International Symposium on Automated Technology for Verification and Analysis (ATVA)*. Volume 5311 of Springer-Verlag *Lecture Notes in Computer Science*, pages 4–17, October 2008.

[53] Joseph R. Shoenfield. *Mathematical Logic*. Addison-Wesley, Reading, MA, 1967.

[54] *Aerospace Recommended Practice (ARP) 4754: Certification Considerations for Highly-Integrated or Complex Aircraft Systems*. Society of Automotive Engineers, November 1996. Also issued as EUROCAE ED-79.

[55] *Report on the incident to Airbus A340-642, registration G-VATL en-route from Hong Kong to London Heathrow on 8 February 2005*. UK Air Investigations Branch, 2007. Available at `http://www.aaib.gov.uk/publications/formal_reports/4_2007_g_vatl.cfm`.

[56] *The Use of Computers in Safety-Critical Applications: Final Report of the Study Group on the Safety of Operational Computer Systems*. UK Health and Safety Commission, 1998. Available at `http://www.hse.gov.uk/nuclear/computers.pdf`.

[57] *Safety Assessment Principles for Nuclear Facilities*. UK Health and Safety Executive, Bootle, UK, 2006 edition, version 1 edition. Available at `http://www.hse.gov.uk/nuclear/saps/saps2006.pdf`.

[58] *Health and Safety at Work etc. Act*. UK Health and Safety Executive, 1974. Available at `http://www.hse.gov.uk/legislation/hswa.htm`; guidance suite at `http://www.hse.gov.uk/risk/theory/alarp.htm`.

[59] *Defence Standard 00-56, Issue 4: Safety Management Requirements for Defence Systems. Part 1: Requirements*. UK Ministry of Defence, June 2007. Available at `http://www.dstan.mod.uk/data/00/056/01000400.pdf`.

[60] Mark Utting and Bruno Legeard. *Practical Model-Based Testing*. Morgan Kaufmann, 2006.