SRI International

Technical Report SRI-CSL-15-01 ● July 2015

The Interpretation and Evaluation of Assurance Cases

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

John Rushby’s research was partially supported by NASA Contract NNL13AA00B
under a subcontract to the Boeing Company.

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

Abstract

Assurance cases are a method for providing assurance for a system by giving an argument to justify a claim about the system, based on evidence about its design, development, and tested behavior.

In comparison with assurance based on guidelines or standards (which essentially specify only the evidence to be produced), the chief novelty in assurance cases is provision of an explicit argument. In principle, this can allow assurance cases to be more finely tuned to the specific circumstances of the system, and more agile than guidelines in adapting to new techniques and applications.

The first part of this report (Chapters 1–4) provides an introduction to assurance cases. Although this material should be accessible to all those with an interest in these topics, the examples focus on software for airborne systems, traditionally assured using the DO-178C guidelines and its predecessors.

The second part (Chapters 5 and 6) considers the criteria, methods, and tools that may be used to evaluate whether an assurance case provides sufficient confidence that a particular system or service is fit for its intended use.

Note: this report is an expanded subset of a NASA report titled "Understanding and Evaluating Assurance Cases" developed by myself with Xidong Xu, Murali Rangarajan, and Thomas L. Weaver of Boeing [111]. It consists of the parts that I wrote, with additional material and new sections on probabilistic topics (e.g., Sections 5.3 and 5.4). There are also numerous small changes of emphasis: the NASA report, rightly, strives to be even-handed, whereas this version reflects my personal opinions. In particular, I am more optimistic about, and supportive of, probabilistic methods. Missing from this version are an erudite discussion of Accident Causation Models by Xidong Xu, and a brief survey of some existing assurance cases; both of those can be found in the NASA report. A more comprehensive survey of existing assurance cases is available in another NASA report [104].

Acknowledgements

It is a pleasure to acknowledge the encouragement and advice of C. Michael Holloway, NASA's technical representative for the project that generated the original version of this report, and of Xidong Xu who managed the project at Boeing. The additional material in this version of the report was funded by SRI International and I am grateful to Pat Lincoln, the Director of CSL, for his support.

I am also grateful for invaluable information on the history and evolution of assurance cases provided by Robin Bloomfield of Adelard and City University, and by Tim Kelly and John McDermid of the University of York, and I also thank them for their information and insight on current practices. Patrick Graydon of NASA supplied several helpful comments and references.

I also benefitted from discussions with researchers and practitioners at recent Dagstuhl and Shonan workshops on software assurance and thank them, together with the organizers for their invitations.

Contents

List of Figures

Executive Summary

We build systems and artifacts to provide benefit, but sometimes they go awry and cause unintended harm. This may be due to some unanticipated circumstance, a flaw in design, or a failure in some component. Accordingly, in addition to the system itself, we must deliver assurance that it will not do harm.

In the limit, it would seem that we must think of all the circumstances the system will encounter and everything that could possibly go wrong or fail, and then provide reasons and evidence for believing that no harm will ensue. The difficulty lies in the words "*all* circumstances it will encounter" and "*everything* that could go wrong." We must explore these potentially infinite spaces with only finite resources (and in general it seems reasonable to expend resources in proportion to risks) so the challenge of assurance is to deliver maximally credible (and retrospectively true) reasons and evidence for believing the system will do no harm, while recognizing that it cannot provide an absolute guarantee.

Early methods of assurance had a retrospective basis: accidents and incidents were investigated, and methods and practices promulgated to prevent their recurrence. Assurance then meant ensuring and documenting that all appropriate methods and practices had been applied. Some industries (e.g., medicine and law) have yet to take these first steps but, when applied diligently (as in building codes, for example), the approach is effective. Its limitation, however, is obvious: it is focused on the past and does nothing to anticipate and eliminate new ways of going wrong—and this becomes a dominant concern as systems become more complex and innovative, and less similar to their predecessors.

Accordingly, methods of assurance in advanced engineering industries have become less prescriptive and more focused on *what* must be accomplished, rather than *how* to do it. In safety-critical software, for example, it is generally required that there should be an orderly development process that results in a hierarchical elaboration of requirements, specifications, and executable code, and some evidence of consistency across these levels. Different industries codify their expected or recommended assurance practices in various standards and guidelines. The current guidelines for software on commercial airplanes, for example, are presented in a document known as DO-178C [99], which describes 71 different "objectives" that must be accomplished for assurance of the most critical software.

There are, however, retrospective and prescriptive elements even to guidelines such as these: they codify best practices and collective wisdom, but these are based on experience with previous systems and methods, and may not anticipate the novel hazards of new systems, nor the potential benefits of new methods for assurance. Guidelines such as DO-178C do allow "alternative methods," but require a rationale that assurance objectives will be satisfied. This is challenging because DO-178C does not provide an explicit rationale for its own methods.

Recent approaches to assurance focus directly on this idea of a rationale: in addition to evidence about the system, we are required provide a rationale that explains *why* this collection of evidence provides adequate assurance. Such a rationale could take many forms, from a narrative description to a point by point examination of alternatives and justification of decisions. The approach that has gained favor presents the rationale in the form of a "structured argument," and the overall approach is then referred to as a (structured) *Assurance Case*.

An assurance case is composed of three elements: a *claim* that states the property to be assured, *evidence* about the design and construction of the system, and a *structured argument* that the evidence is sufficient to establish the claim. The structured argument is a hierarchical collection of individual argument steps, each of which justifies a local claim on the basis of evidence and/or lower-level subclaims. A simple example is shown to the right, where a claim **C** is justified by an argument step $\mathbf{AS}_1$ on the basis of evidence $\mathbf{E}_1$ and subclaim $\mathbf{SC}_1$, which itself is justified by argument step $\mathbf{AS}_2$ on the basis of evidence $\mathbf{E}_2$ and $\mathbf{E}_3$. (Note that a structured argument may not be a tree because one subclaim or item of evidence could support more than one argument step.)

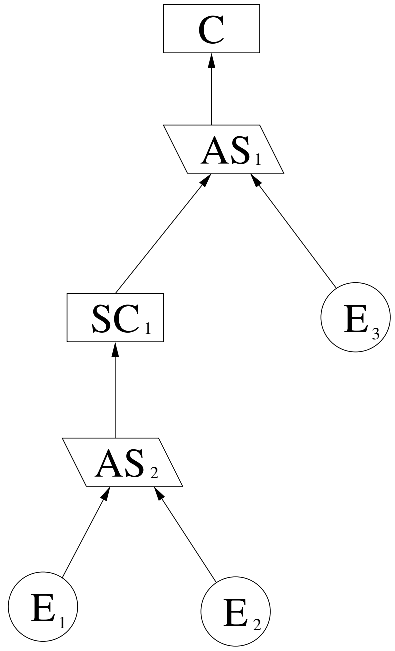


Figure 1: Structured Argument

The question then arises: how do we know that a given assurance case is sound? The same question can be asked of guidelines such as DO-178C and one answer is that these are validated by historical experience: modern airplanes are extraordinarily safe and no serious airplane incident has been traced to faulty software (some have been traced to faulty requirements for systems implemented in software, but that is outside the remit of DO-178C). But one reason for looking beyond guidelines and toward assurance cases is to admit new methods of assurance (e.g., static analysis) and new kinds of systems (e.g., closer integration of air and ground elements in NextGen), so relevance of the historical record becomes unclear. Another answer to the effectiveness of DO-178C is that it implicitly incorporates a sound assurance case: the underlying argument is not made explicit but surely informed the committee deliberations that produced it. A useful way to examine the relationship between guidelines and assurance cases is to reconstruct the assurance case implicit in selected guidelines. Michael Holloway has done this for DO-178C [63] and in this report we undertake a similar but more localized and discursive exercise as a way to introduce ideas and issues concerning assurance cases to those who have some familiarity with existing guidelines.

In particular, we examine Section 6.3.1 of DO-178C, which is concerned with "Reviews and Analyses of High-Level Requirements." We suggest that the purpose of these reviews and analyses is to establish the claim that the High-Level Requirements (HLR) for the software correctly represent the System Requirements (SR). The basic argument strategy is to establish that everything in the SR is correctly specified in the HLR (i.e., nothing is left out, and what is included is correct), and everything in the HLR is required by the SR (i.e., nothing extraneous is included). We find objectives in DO-178C Section 6.3.1 that correspond to each of these two subclaims, but there are also five other objectives—and we ask what is their purpose?

We have already noted that assurance cannot provide absolute guarantees, and in assurance cases this becomes manifest in the construction and evaluation of arguments, where we expect each step to strongly justify its claim but cannot expect unequivocal "proof." Such less-than-definitive argument steps are said to be *inductive* (in contrast to unequivocal *deductive* steps). One way to buttress support for an inductively justified claim could be to provide additional subclaims or evidence that strengthen some aspects of the justification, even though they do not change its inductive character. These intended strengthenings are called *confidence claims* and some of the five "additional" objectives of DO-178C Section 6.3.1 can be interpreted in this way. For example, the objective "HLR are Verifiable" can be interpreted as a quality measure on the HLR that increases our confidence that the primary objectives concerning consistency of SR and HLR can be performed correctly.

The report considers several ways of organizing an argument around DO-178C Section 6.3.1 and uses these to examine topics in the construction and interpretation of assurance case arguments. The top claim in a software assurance case is generally correctness (of executable object code with respect to system requirements), so we also examine a simple system assurance case, where the top claim is safety (or, in general, absence of some specific harm).

Assumptions are an example of a topic that arises in both cases: in a system case, for example, we may establish safety of some element by arguing that it is safe in each of its operating "modes": air, ground, and in transition between these. But then we need an assumption that there are no other modes and we have to consider whether subclaims that establish assumptions should be treated differently than other subclaims—it seems plausible that they should because other subclaims in an argument step may not "make sense" unless the assumption is true. (A simple example is that $\frac{x}{y} > 3$ does not "make sense" unless $y \neq 0$.) We demonstrate that since truth of a claim requires all its supporting subclaims to be true, there is little difference between assumptions and other subclaims, and we could eliminate the difference by stipulating that each subclaim is interpreted on the supposition that all the other subclaims in its argument step are true. But this could lead to circularity; a sound compromise is to stipulate that that each subclaim is interpreted on the supposition that subclaims appearing earlier in its argument step are true. These issues concerning assumptions exemplify a general weakness we find in formulations and notations for assurance case arguments: that is, a lack of rigorous semantics.

An assurance case and its argument can be presented in several ways: for example, as structured text, in a formal notation, or in graphical form. Two methodologies, each with a graphical notation, are widely used: these are CAE (Claims-Argument-Evidence, developed by Adelard) and GSN (Goal Structuring Notation, developed at the University of York). The two notations are similar and differ primarily in the shapes of their graphical elements and the direction of their arrows: the example in Figure 1 uses the element shapes from GSN, but its arrows are reversed, as in CAE. Where the approaches differ more substantially is in the focus of their methodologies: CAE places more stress on the justification for arguments (so the text associated with the argument steps $\mathbf{AS}_1$ and $\mathbf{AS}_2$ in Figure 1 could be quite lengthy), while GSN is more focused on structure (so the overall thrust of an argument might be conveyed by its diagram).

Both methodologies provide guidance and suggested "outlines" for the construction of good arguments. For CAE, these are called *blocks* and focus on ways to construct individual argument steps. One way is to decompose a claim into subclaims (and/or evidence) each of which addresses one element of the decomposition; an example is the decomposition over "modes" (air, ground, and transition) mentioned earlier. The CAE methodology identifies many different forms of decomposition (e.g., by component, property, environment, configuration) as well as other kinds of building blocks, together with the assumptions (or "side conditions") required for them to be valid. Whereas CAE blocks focus on the individual steps of an argument, GSN provides "patterns" that address complete arguments (or subarguments for large arguments) of various stereotypical forms. Because they address larger argument fragments, it is more difficult to state the assumptions under which templates are valid than it is for blocks, so guidance for GSN patterns is given in the form of “critical questions” that should be considered when adopting and instantiating a template.

Several tools are available to support various aspects of assurance case development and review. All can render an assurance case as a GSN diagram, and one (the most widely used) also supports CAE. Most also support graphical editing of the case. The capabilities of the tools that seem most widely used are described in the report, and others are outlined. Many tools support an interchange format for assurance cases called SACM that has been standardized by the Object Management Group (OMG).

In our opinion, a weakness in both CAE and GSN, and an impediment to trustworthy application of assurance cases, is that some aspects of their semantics are undefined, or are defined by the particular tool employed. One example is the interpretation of assumptions described earlier. Another is support for modular development of large arguments. It is impractical for one individual to develop and manage a large assurance case: there must be some way to approach it in a modular fashion so that those working on one part of the argument need not be concerned with internal details of another part, yet coherence of the overall case must be ensured. GSN does provide constructs for modularity (e.g., an “away goal” is a reference to a subclaim defined elsewhere) but we are not convinced that all necessary details are attended to. For example, it is critical that those who define a subclaim and those who use it have the same interpretation of its intended meaning and of the context in which it may be employed. Context is defined by assumptions so these should be stated at the point of definition and enforced or checked at the point of use. GSN does provide general (i.e., not specifically modular) assumption and justification nodes, and also context nodes, but the exact semantics and intended interpretation for these are not well defined (and, in the case of context nodes, are currently subject to debate) and there are no tool-supported checks. Modularity in CAE is built on that of GSN, so the same concerns apply.

Although the general ideas of assurance cases and structured arguments are straightforward, safety is all about attention to detail, so we would wish to see languages and tools that provide clear and comprehensive semantics for the assurance cases described with their aid. Graphical presentation, support for browsing large cases, and integration with system development tools, are all highly desirable capabilities, but secondary to precise and unambiguous interpretation for the case that has been described.

Next, the report moves on to the evaluation of assurance cases: how to determine if an assurance case is sound, credible, and sufficiently strong to justify deployment of the system concerned. There are really two separate issues here. An assurance case cannot provide unequivocal proof; it is an inductive argument and there will inevitably be some uncertainty about some of its elements. The first issue is how to assess the impact for the overall argument of acknowledged doubt in some of its elements: we need to be able to tell when the overall argument delivers adequate assurance, and when some of its steps need to be strengthened with different or additional evidence or a changed rationale. The second issue is the fallibility of human judgment in performing these assessments: primarily, this means finding ways to counter the human tendency for "confirmation bias."

A fundamental difficulty is that there is no really satisfactory nor generally agreed approach for assessing inductive arguments. However, the report does explore some of the candidates. For deductive arguments, logic provides the criterion for evaluation: a deductively sound argument is a *proof* in some logical system, and we use this as a benchmark in considering methods for assessing inductive arguments.

An approach developed by Steven Toulmin in the 1950s influenced some of the pioneers in assurance cases and continues to be widely referenced. We regard this as unfortunate. A principle of logic is that the reasoning and the subject matter of an argument can be treated separately: if I know that A implies B and I know A, then I can conclude B independently of the meanings of whatever are substituted for A and B. More generally, if we agree on the premises to a deductive argument and a conclusion follows by the rules of logic, then we have to accept that conclusion (furthermore, if the premises are true statements about the world, then so is the conclusion). Toulmin was concerned with arguments in highly contested areas such as aesthetics or ethics where participants might not agree on premises, and his approach sacrifices the separation of reasoning from subject matter that is the hallmark of logic. We think that assurance cases may need to adjust the ideas of classical logic to accommodate inductive arguments, but should not abandon them.

A completely different approach to the evaluation of assurance case arguments is to interpret them probabilistically rather than logically. Methods such as Bayesian Belief Networks (BBNs) or the Dempster-Shafer theory of evidence can be used to represent causal or conditional relationships among the evidence in a case and then allow the probability of a claim to be calculated from that of the evidence supporting it. One objection to this approach is that it can be very difficult to assess credible probabilities for many items of evidence, and another is that it ignores the rationale provided by the argument and just looks at the evidence.

There are several proposals that strive to combine probability and logic to yield "probability logics" but, in our opinion, none are fully satisfactory. Nonetheless, the approach we advocate does employ such a combination, but in a very simple form that is tailored to the character of assurance case arguments.

First, we note that by introducing additional subclaims if necessary, it is straightforward to convert an argument into a *simple form* where each argument step is supported either by subclaims or by evidence, but not by a combination of the two. In Figure 1, for example, step $\mathbf{AS}_2$ is supported by evidence alone, but $\mathbf{AS}_1$ is supported by a combination of evidence ($\mathbf{E}_1$) and subclaims ($\mathbf{SC}_1$) and is therefore not in simple form; it can be converted to simple form by introducing a new subclaim and argument step above $\mathbf{E}_1$ as shown in Figure 2. Argument steps supported by subclaims are called *reasoning steps*, while those supported by evidence are called *evidential steps*; the two kinds of step are interpreted differently.

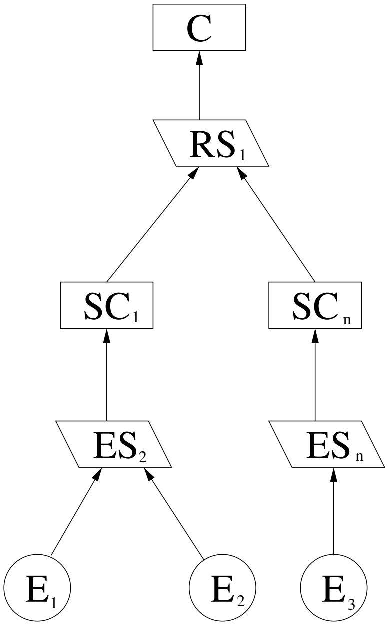


Figure 2: Argument in Simple Form

Evidential steps are interpreted *epistemically*: they are the bridge between our concepts (expressed as subclaims) and our knowledge of the world (recorded as evidence). Informally, the combination of evidence supplied in the step (which may include confidence items) is "weighed in the balance" to determine whether it crosses some threshold that allows the subclaim to be treated as a "settled fact." More formally, this process can be framed in terms of probabilities and undertaken with BBNs using ideas from Bayesian Epistemology. Graduated assurance, where stronger or weaker—but related—arguments are used for elements that pose different degrees of risk (as in the Software Levels of DO-178C), can be accommodated by raising or lowering the bar on the "weight" required for evidential steps.

Reasoning steps are interpreted *logically*: we must determine if the conjunction of subclaims in a step deductively entails its claim. A radical element here is the requirement that this entailment be deductive, with a consequent side effect that confidence items cannot be used in reasoning steps.

Our rationale for this is the need to confront the second issue mentioned earlier: the fallibility of human judgment and its tendency to confirmation bias. By requiring reasoning steps to be deductive, we make it very clear what the evaluation of these steps must accomplish. In contrast, there is no clear criterion for evaluating inductive reasoning steps and a consequent temptation to add confidence items "just in case," thereby complicating the argument and its evaluation for an uncertain benefit.

Confirmation bias is the human tendency to seek information that will confirm a hypothesis, rather than refute it. The most effective counterbalance to this and other fallibilities of human judgment is to subject assurance cases to vigorous examination by multiple reviewers with different points of view. Tools can assist this process by facilitating browsing and exploration of a case and by recording what has been examined and any comments made. More challenging reviews could entail active probing of a case and "what-if" explorations. Tools can assist this if the reasoning steps are supported by theorem proving (which is feasible if these steps are deductive) and the report outlines the ideas of defeasible reasoning, which can be used to "add up" the consequences of conflicting or inconsistent opinions. Defeasible reasoning can be valuable in time-constrained contexts where it is necessary to use whatever information is available, but for assurance cases we think it is essential to resolve conflicting opinions and to achieve consensus on the true state of affairs, rather than to somehow "add up" the differences. However, defeasible reasoning provides the useful concept of a *defeater*; in an assurance case this would be a reason for doubting that the subclaims to a reasoning step really do entail its claim, or that the evidence cited in an evidential step has adequate "weight." Defeaters to an argument are rather like hazards to a system; thus, a systematic and potentially effective way to review assurance case arguments is by proposing plausible defeaters for each argument step and checking that the argument resists each challenge.

We conclude that assurance cases are a natural, attractive, and potentially effective evolution in methods of assurance. They can be adapted more readily than standards and guidelines for new kinds of systems and new technologies, and they can allow more customized allocation of effort. We find some issues in popular notations and tools for assurance case arguments, and in the foundations of inductive arguments, but prefer this state of affairs to that of current standards and guidelines where similar issues are masked by absence of an explicit argument. The central issue in all forms of inductive arguments is identifying and managing sources of doubt or uncertainty; we recommend that uncertainty should be restricted to evidential argument steps and that the reasoning (or interior) steps of an argument should be deductive. This is a radical proposal and it remains to be seen whether it is feasible in practice.

Our largest concern is the degree of independent review that can be applied to a bespoke assurance case. Conscious search for defeaters to the argument (rather like hazards to a system) could provide a systematic means of evaluation, but the number of reviews and reviewers will be fewer than for community-endorsed guidelines. We therefore advocate hybrid approaches. In particular, we recommend that future revisions to guidelines such as DO-178C should be structured as assurance case templates, and that most assurance cases should be constructed by instantiating and customizing community-endorsed templates, rather than as fully bespoke developments.

Chapter 1

Introduction

Assurance Cases are a relatively new approach to ensuring safety and other critical properties, such as security, for complex systems. As systems have become more complex, so it has become increasingly difficult for standards to specify how to make them safe. The context, requirements, design, and implementation of each system are sufficiently distinctive that its safety requires unique consideration. Accordingly, standards and guidelines for safety evolved from prescriptions on how systems should be built (e.g., rivet holes should be drilled, not punched), to guidelines on the kinds of reviews and analyses that should be performed, and the evidence that should be collected from these. Assurance cases take this a step further and give the developer freedom to select the analyses that will be performed and the evidence that will be collected, but require a rationale that justifies the choices made and "makes the case" that these ensure safety, or other critical property.

The rationale could take many forms and be presented in many ways, but the approach that has gained favor is that of a structured argument. Such an argument justifies a claim (about safety or other properties) on the basis of lower-level subclaims, which are themselves justified on the basis of still lower-level subclaims and so on, until we reach subclaims that are justified on the basis of observed and measured evidence. Thus, an assurance case provides a *structured argument* that justifies *claims* about a system on the basis of *evidence* about its design, implementation, and other attributes.

This report is about understanding and evaluating assurance cases and is primarily addressed to those concerned with assurance and certification of software for civil airplanes, but should be accessible to those interested in assurance for almost any kind of system. Civil airplanes are extraordinarily safe and it is reasonable to conclude that existing guidelines and practices are effective and should not be be changed gratuitously. However, the nature and context of airborne software is changing (e.g., much closer integration with ground systems in NextGen, greater autonomy, and unmanned vehicles) and techniques for analysis and implementation are evolving (e.g., ubiquitous static analysis, automated synthesis, and adaptive systems), so some adjustment seems inevitable. And although present practices are undeniably effective, we do not really know *why* they are effective. So one immediately useful application of assurance cases would be to document the rationale for existing guidelines and standards prior to making any changes (Michael Holloway is documenting DO-178C in this way [63]).

However, the main reason for interest in assurance cases is that they seem to be the way forward: they are the required or preferred method for assurance in several industries and the idea of a rationale for safety, documented as an argument, has strong intellectual appeal. The danger is that this appeal may be specious: the idea of an argument is attractive, but human reasoning is fallible and may be incapable of assessing large arguments (and assurance case arguments are typically huge) in a reliable manner; furthermore, we need to be sure that all stakeholders and participants interpret an assurance case argument in the same way, so we should have a clear and agreed semantics for them. This last is difficult because no method of assurance can provide unequivocal guarantees of safety, so an assurance case argument is *inductive*—that is, it strongly suggests but does not guarantee its claim—and there is no generally agreed semantics for inductive arguments. Hence, a large part of this report is concerned with topics concerning the formulation and interpretation of inductive arguments.

The structure of this report is the following. The next chapter provides a historical overview of the evolution of methods of assurance, paying some particular attention to assurance for airplanes and their software, and concluding with safety cases, structured safety cases and, finally, assurance cases.

Chapter 3 introduces the ideas of assurance cases by developing several assurance case fragments around topics in Section 6.3.1 and Table A-3 of DO-178C, the guidelines for airborne software [99]. Some of the discussion involves mind-numbing detail, but assurance is all about attention to detail so we consider this examination justified and necessary. Because software and system assurance cases have somewhat different concerns, we then perform a similar examination of a simple system assurance case.

Chapter 4 looks at notations and tools for assurance cases. There are two widely-used graphical notations for assurance cases (CAE and GSN) and we describe these and tools that support them. The basic notion that an assurance case argument consists of individual steps that justify claims on the basis of evidence or lower-level subclaims becomes more complicated when notational constructions needed to manage large cases are added. These include support for modularity, contexts, assumptions, and collections of various kinds of assurance case fragments or outlines that can guide development of a new case.

Chapter 5 considers how to evaluate the soundness and strength of an assurance case and how to determine whether it provides sufficient confidence to deploy the system concerned. There are both technical and human factors here. Technical factors concern semantics: that is, the meaning ascribed to an assurance case. We review the ideas of classical logic, since these underpin "deductive" arguments. However, assurance case arguments are often inductive rather than deductive (i.e., they strongly suggest their claim is true, but cannot prove it) and so we examine alternatives to classical logic such as Toulmin's approach, and those that combine logic and probability.

Human factors in assurance case evaluation concern fallibilities of human reasoning and the danger of "confirmation bias," which is the tendency to seek information that confirms a hypothesis, rather than challenges it. The most effective way to counteract these seems to be the "wisdom of crowds," that is the scrutiny of many reviewers having different points of view. The academic topics of argumentation, dialectics, and defeasible logic contribute techniques for probing and resolving contested arguments and we review some of these ideas. In particular, we describe the idea of a "defeater." A defeater to an argument is rather like a hazard to a system: that is, a reason why it might go wrong. A systematic search for plausible defeaters may be an effective way to probe an assurance case and counteract the influence of confirmation bias.

Finally, Chapter 6 presents our conclusions. We are broadly supportive of the aims and methods of assurance cases but critical of some aspects of their notations and of the lack of agreed semantics for inductive arguments. We propose such a semantics in which the evidential leaves of an argument are "weighed" (which can be formalized using ideas from Bayesian Epistemology) to ensure that support for their claim exceeds some threshold, and the interior reasoning steps of the argument are interpreted in classical, deductive logic. This locates all uncertainty in the evidential steps; the interior steps are like a proof. This proposal raises the bar on the interior part of an assurance case argument, which in current practice is expected to be merely inductive. It remains to be seen if it is feasible to apply the proposal in practice, and whether it finds acceptance.

Our main reservations concern the trustworthiness of fully custom (i.e., "bespoke") assurance cases that are likely to receive little independent review. Apart from its developers, such a case may be reviewed only by those responsible for regulation or certification of the system concerned. Despite responsible diligence, this may be insufficient to overcome the propensity for confirmation bias. Accordingly, we recommend that the outline structure of an assurance case should be derived from intensively scrutinized community-endorsed templates that distill best practice and lessons of history, rather in the way that guidelines and standards do today.

Chapter 2

Assurance Cases in a Historical Perspective

Safety has been a consideration in the design and construction of human artifacts since their very beginning. The earliest regulations, from Babylon in about 1772 BC, focused on product liability and appropriate penalties for failure.

> "If a builder build a house for some one, and does not construct it properly, and the house which he built fall in and kill its owner, then that builder shall be put to death" [52, Section 229].

Regulations and methods for safety assurance have changed over the years and we describe some of their development in the following sections. We begin in Section 2.1, with an outline of their evolution during the 19th century then, in Section 2.2, describe recent and current regulations and methods for safety assurance in civil aviation, particularly airborne software. This report is primarily written for those concerned with airborne software and some of its material (particularly Chapter 3) assumes some familiarity with practices in this field, so Section 2.2 may serve to introduce these practices and their terminology to readers with different backgrounds. The final two sections of this chapter, 2.3 and 2.4, outline the development of safety cases and their evolution into structured safety cases and assurance cases.

2.1 Early Industrial Regulations and Codes

From their ancient beginnings, civil engineering projects, buildings, and ships remained the largest and most complex engineered products for the next 3,500 years, up until the beginnings of the Industrial Revolution. With the rise of factories, workplace safety became a concern, although the motivation was often the value of lost goods rather than human lives.

> "It is said that Napoleon, lacking gunpowder due to a disastrous series of explosions in his gunpowder factories, decreed that factory owners and their families should live in their factories" [106].

As the industrial revolution progressed, the safety of its new technologies became a concern and attention began to be paid to learning the causes and preventing the occurrence of failures. This was particularly so with high pressure steam engines, whose boilers were prone to explode. From 1830 to 1837, the United States government partially defrayed the costs of research by the Franklin Institute to determine the causes of steam boiler explosions. This led to a series of reports that developed some of the relevant science and provided guidelines for the design, construction and operation of steam boilers, together with recommendations for regulation. Largely in further response to concerns for boiler safety, the Institution of Mechanical Engineers (IME) was formed in the UK in 1847, and the American Society of Mechanical Engineers (ASME) in the USA in 1880. These societies emphasized the importance of specialized mechanical knowledge and set about further developing and codifying this knowledge, and establishing standards.

Due to opposition in political and business quarters, government regulation lagged these developments. But in the USA, the Steamboat Act of 1852 introduced requirements for hydrostatic testing of boilers and installation of a safety valve. The act further required that both pilots and engineers be licensed by local inspectors. An assessment published in 1899 of the comparable UK acts noted that maximum pressures in boilers used for manufacturing had increased over the previous 15 years from 80 pounds per square inch to 200 and, in exceptional cases, to 250 or 300 pounds per square inch. Presumably, there were also many more boilers in use than previously. Yet the number of explosions in the 10 years from 1886 to 1895 was approximately half that in the 10 years from 1866 to 1875 (317 vs. 561) and the numbers of persons killed was reduced to less than a third (185 vs. 636) [59]. These improvements were attributed to better design and operation, and better technology such as use of drilled rather than punched holes for rivets.

We can note that these early laws mostly focused on establishing regimes (often reinforced by the insurance industry) for inspections and investigations, and these stimulated the general development and promulgation of know-how. But the laws also identified specific hazards and required specific means of assurance and mitigation (e.g., hydrostatic testing, and release valves). Similar developments took place in other industries. For example, following an accident at Hartley Colliery in 1862 where 204 miners suffocated underground when the beam of a pumping engine broke and blocked the only mineshaft and means of ventilation, the UK introduced legislation requiring that every seam in a mine should have at least two shafts or outlets [24]. Pennsylvania introduced a similar law in 1869. Here again, we see a specific hazard and means of mitigation written into law.

Later, however, legislation shifted from such specific mandates toward recognition of the standards and "codes" being developed by professional organizations. For example, in 1907, following several deadly boiler explosions, the State of Massachusetts passed a law that imposed rules based on ASME's evolving Boiler Code. The first completed version of the ASME code, which was issued in 1914, was much more comprehensive than that written into the 1907 law and formed the basis for laws in other states.[1] Over time, the ASME code has developed to comprise more than 16,000 pages in 28 volumes.

2.2 Civil Aviation Regulations and Guidelines

Safety in aviation built on these prior developments in other industries. First, we should note that although the Wright Brothers were, on 17 December 1903, the first to achieve controlled flight, their patent dispute with Glenn Curtiss retarded US aviation at a time when Europe, stimulated by the threat and actuality of war, was making significant advances. Accordingly, on 3 March 1915, the US congress established the National Advisory Commission on Aeronautics (NACA), the predecessor of NASA, which is the sponsor of this report [8].

By the 1920s, a nascent commercial airline industry was developing, but costs were high and so were perceived and real risks, and paying passengers were rather few. To encourage commercial aviation, the Contract Air Mail Act of 1925 authorized the United States Post Office to contract with private airlines to provide feeder routes into the main transcontinental air mail routes operated by the Post Office itself. This stimulated traffic, but aircraft safety, air traffic control (very rudimentary at that time), and navigation aids remained the responsibility of the private aircraft manufacturers, airlines, and airports. These parties urged government development of infrastructure and regulation.

Accordingly, the Air Commerce Act of 1926 charged the Secretary of Commerce with fostering air commerce, issuing and enforcing air traffic rules, licensing pilots, certifying aircraft, establishing airways, and operating and maintaining aids to air navigation. These responsibilities of the Department of Commerce were assigned to a newly created Aeronautics Branch, which subsequently evolved, through many reorganizations, renamings, and changes in governing legislation, into the Federal Aviation Administration (FAA) in 1958. The precursors to the National Transportation Safety Board (NTSB), which is charged with investigating the causes of aircraft accidents, were also established by the 1926 act.

Aviation in the USA is now regulated under the Federal Aviation Regulations (FARs), which are part of Title 14 of the Code of Federal Regulations (CFRs). (The

[1] Even today, not all states have laws on boiler safety; South Carolina passed legislation only in 2005 (without the governor's signature).

abbreviation FAR is also used for the Federal Acquisition Regulations, which are Title 48 of the CFRs, but will be used here only in the aviation sense.) The various Parts of the FARs are grouped into subchapters, of which we are mostly interested in certification requirements for aircraft, found in Subchapter C and, within that, the sections that are interpreted to apply to software; however increasing integration between on-board and ground systems under NextGen means that airborne software may also be subject to air traffic control and flight rules, which are in Subchapter F. Within Subchapter C, the airworthiness regulations for transport aircraft constitute Part 25 of the FARs; regulations for smaller aircraft constitute part 23 and those for normal and transport rotorcraft are found in Parts 27 and 29, respectively.

The FARs are terse; interpretation of the regulations and descriptions of "acceptable means of compliance" are generally issued as Advisory Circulars of the FAA. In particular, software is not mentioned in the FARs; Section 25.1309, titled "System Design and Analysis" and less than a page in length, states requirements on "systems" that are interpreted to flow down as the governing regulations for software assurance. These are elaborated somewhat in AC 25.1309, which is the advisory circular corresponding to FAA 25.1309. The European Aviation Safety Agency (EASA) Certification Specifications CS 25 are largely harmonized with FAR 25, and its acceptable means of compliance are collected as AMC 25.[2] CS 25.1309 and AMC 25.1309 are the EASA equivalents of FAR 25.1309 and AC 25.1309, respectively.

The essence of FAR 25.1309 is that system failures that could have really bad consequences must be very rare, and no single failure should be able to cause the worst ("catastrophic") consequences. AC 25.1309 elaborates these to require an inverse relationship between the probability and severity of failure conditions and provides definitions for the various severities and their acceptable probability. Most critically, "catastrophic" failure conditions are those "which could prevent continued safe flight and landing" and these must be "extremely improbable" which means they are "so unlikely that they are not anticipated to occur during the entire operational life of all airplanes of one type."

When a new aircraft type is submitted for certification, the certification authority (in the United States, this is the FAA), in consultation with the applicant (i.e., the airframe manufacturer), establishes the *certification basis*, which defines the applicable regulations together with any special conditions that are to be imposed.[3] The applicant then proposes a *means of compliance* that defines how development of the aircraft and its systems will satisfy the certification basis.

For some aspects of software, industry standards are recognized as acceptable means of compliance for FAR 25.1309. For example, AC 20-174 "recognizes the Society of Automotive Engineers (SAE) Aerospace Recommended Practice (ARP)

[2] CS 25 and AMC 25 are issued as separate books within a single document [38].

[3] An example of a special condition is one for the Boeing 787 concerning protection of its system and data networks [41].

4754A, Guidelines for Development of Civil Aircraft and Systems, dated December 21, 2010, as an acceptable method for establishing a development assurance process," and AC 20-115C "recognizes. . . RTCA DO-178C, Software Considerations in Airborne Systems and Equipment Certification, dated December 13, 2011. . . as an acceptable means of compliance for the software aspects of type certification."

SAE, mentioned above in connection with ARP 4754A, was founded in 1905 as the Society of Automobile Engineers, but generalized its remit and name in 1916 to the Society of Automotive Engineers and now covers both aerospace and automobile engineering. RTCA, mentioned above in connection with DO-178B (DO stands for Document Order), was founded in 1935 as the Radio Technical Commission for Aeronautics and is used as a Federal advisory committee, meaning that in response to requests from the FAA it establishes committees to develop recommendations and guidelines for the federal government. RTCA states that "our deliberations are open to the public and our products are developed by aviation community volunteers functioning in a consensus-based, collaborative, peer-reviewed environment." EUROCAE, founded in 1992 as the European Organization for Civil Aviation Equipement, serves EASA as the European counterpart to RTCA. EUROCAE and RTCA generally establish parallel committees that meet jointly and issue parallel documents: for example, EUROCAE ED-12C (ED stands for EUROCAE Document) is the same as RTCA DO-178C.

In later sections of this report, we will examine part of DO-178C in some detail and discuss its relationship to assurance cases. Here, however, we briefly recount the evolution of DO-178C from earlier versions of the guidelines.

Johnson [71] states that aircraft software was originally seen as an adjunct to mechanical and analog systems and was assessed for safety in the same way as those systems. That is, failures were attributed to components and the reliability of components was established by statistical testing. But during system certifications in the late 1970's, it became evident that software was achieving sufficient complexity that design errors should be a concern.

The original DO-178, which was issued in 1982, aimed to document "best practices" for establishing that software is safe and does not contribute to the system hazards. It allowed that a system's software development rigor could vary by the system failure severity: a system could be categorized as critical, essential or non-essential. DO-178 also established the need for a certification plan that included software aspects. Johnson states that DO-178 was written at a "conceptual" level and that compliance was achieved by meeting its "intent."

DO-178A, which was issued in 1985, was quite different to the original DO-178 and built on lessons learned with that document. Its purpose was to establish techniques for orderly software development with the intent that their application would produce software that is documented, traceable, testable, and maintainable. Three software "levels" were established, with the greatest assurance effort required for Level 1 and the least for Level 3; these levels were to be determined by considering the possible effects of malfunctions or design errors, with adjustment according to the system design and implementation techniques (thus, software in an "essential" system need not all be at Level 1).

Johnson [71] states:

> "Strengths and weaknesses of DO-178A soon became apparent. Literal interpretation, particularly from diagrams were a problem. Also, necessary certification submittal items were frequently contended despite the fact that they were subject to negotiation with the regulatory agencies and could vary between agencies. Misuse included non-allowance of life cycles other than the traditional waterfall model and non-existence of documents not identified as 'required.' Contention also arose on the required certification effort.
>
> "In general, the knowledge of why the certification requirements existed and the purpose of the requirements failed to be understood or appreciated."

The next revision, DO-178B, appeared in 1993. According to Johnson [71], a major motivation was to reduce dependence on the few software certification experts by documenting practice and policy. The failure categories were increased from three to five, and the related software levels were likewise increased to five and renamed Level A (the highest) through Levels B, C, D, to E.

DO-178B does not specify how software development and assurance should be performed, but it does specify that these should include certain activities, such as reviews and testing, should produce certain documents, such as plans for various aspects of development and assurance, descriptions of requirements and designs and so on, and that there must be strong configuration management that includes traceability from requirements to code and tests. In all, DO-178B describes 66 "objectives" of this kind and requires that all of them must be applied to Level A software, 65 of them to Level B, 57 to Level C, and 28 to Level D. Furthermore, at each level, it requires that some of the objectives are performed "with independence" from the development team.

DO-178B was used quite extensively but there was substantial evolution in software development and assurance practices in the years following its introduction. These included object-oriented methods and languages, model-based software engineering, formal methods and so on. In principle, these could be accommodated under DO-178B because its Section 12.3 allows the use of "alternative methods." The most challenging of the guidance items for such methods is the requirement for a "*rationale for use of the alternative method which shows that the system safety objectives are satisfied*" [98, Section 12.3.b(3)]. This is challenging because DO-178B does not provide an explicit rationale for its own methods.

Accordingly, a committee was established to update DO-178B; following more than five years of deliberation, DO-178C was issued in 2011 [99]. This revision makes few substantive modifications to the text of DO-178B, but adds four supplements that address "Software Tool Qualification Considerations" [100], "Model-Based Development and Verification" [101], "Object-Oriented Technology and Related Techniques" [102], and "Formal Methods" [103]. One notable modification to DO-178B occurs in Section 12.3, where DO-178C adds "*one technique for presenting the rational for using an alternative method is an assurance case, in which arguments are explicitly given to link the evidence to the claims of compliance with the system safety objectives.*" We now turn to the origins of safety and assurance cases.

2.3 Safety Cases

The approach to assuring safety that eventually developed into the notion of an assurance case had its origins in the basic notion of a "safety case" or "safety report," whose precursors had developed in several industries, but were given impetus by the report in 1972 of a committee appointed by the UK government to consider the broad topic of health and safety law [105]. This report, often referred to as "The Robens Report" after its chairman, Lord Robens, recommended a rather radical change in the approach to safety. First, it argued that those who create risks are responsible for controlling them: this points away from government mandates and toward some element of self-regulation. Related to this was the idea that regulations should set goals, rather than prescribe methods and solutions. Second, it argued that standards for safety should be "reasonably practicable," that is, controls should be commensurate with risk and cost must be taken into account. The Robens Report laid the foundation for subsequent UK legislation (The Health and Safety at Work Act, 1974) and set the tone for much future thinking on the topic.

The Robens Report was not prompted by any recent disasters but shortly afterwards, in 1974, a major accident at a chemical plant in Flixborough killed 28 people and injured many more. The newly-created UK Health and Safety Executive drafted a set of regulations that required hazardous installations to develop what, in essence, would be a safety case. But before those regulations could be enacted there was another major accident, at Seveso in Italy, that released a large quantity of dioxin into the atmosphere. The European Community issued a directive that was based on the prior British work and required production of a "safety report" demonstrating adequate consideration of dangerous substances, potential accidents and provision of effective safety management systems.

Then, in 1988, an explosion on the Piper Alpha North Sea oil production platform killed 167 people. Despite the 1974 legislation, safety in the offshore oil and gas industry was still governed by a pre-Robens "Permit To Work" approach. The inquiry into this disaster, conducted by Lord Cullen [23], reviewed and endorsed the approach of the Robens Report and recommended adoption of goal-based safety cases. This was done and, within a decade, reportable offshore industry accidents in the UK had declined by more than 75%.

It should be noted that all the accidents just described were in process industries and the notion of safety case that was promulgated by the various reports, regulations, and legislation was largely driven by experience in those and similar industries. The purpose of the shift from prescriptive approaches to safety cases was to eliminate a “check the box” mentality on the part of “duty holders” and to foster a safety culture in which they took responsibility for understanding and controlling hazards.

The details of what constituted a safety case varied across the different industries and regulation, but would generally include the following topics [68].

1. The system or activity being addressed, together with details of its scope, context or environment.

2. The management system used to ensure safety.

3. Evidence that risks have been identified and appropriately controlled, and that the residual level of risk is acceptable.

4. The requirements, legislation, standards and policies applicable, with evidence that they have been met or complied with.

5. Independent assurance that evidence presented is sufficient for the application in question.

The management system (item 2 above) is often the main safety concern in process industries, although design of the plant is obviously also important. In other industries, the rôles are reversed and design is the dominant issue. For example, the UK nuclear power industry has been using safety cases for design since the 1950s (following the Windscale accident in 1957) and was required to do so by legislation introduced in 1965. The essence of a safety case for the design of a nuclear plant would be a “narrative” about the design and its rationale. In the USA, similar documents were required as a “Pre-Construction Safety Report” (PCSR). As designs became more complex, limitations became apparent in the narrative style of safety case presentation, and this led to the development of more structured styles.

2.4 Structured Safety Cases

In the late 1970s, the UK was considering switching from its indigenous Advanced Gas Cooled Reactor design (whose construction was plagued by overruns and delays) to the US Pressurized Water Reactor design, which employed more computer-based functions, including the safety-critical shutdown system. Engineers working on the UK safety case realized they did not have a good way to incorporate computer systems into the case. To resolve this, they developed the idea of a "structured safety case," whose key innovation was to require that there be an *argument* to explain how the evidence in the case ensured satisfaction of the goals.

This approach gained some acceptance in the nuclear industry in Europe and, more extensively, in the UK, where it was incorporated in a number of policy initiatives through the 1980 and 1990s [9]. The generic term *argument* embraces a wide range of rigor in organization, presentation, and overall coherence. Despite exhortations that safety case arguments should be "convincing and valid," they were typically presented in linear prose that was difficult to review. Writing in 1995, Wilson, Kelly and McDermid [130] reported

> "In studying industrial safety cases we have invariably found it necessary to annotate and cross reference them, in order to 'pull out' the structure and argument. Often the result was the discovery of a convincing argument—but the onus on discovering the argument was on the reader."

Accordingly, the interpretation of "structured safety case" was sharpened during the 1990s to require that the argument itself should be "structured" [9]. Some of those working on these topics drew on the work of Stephen Toulmin [122] and on techniques for "diagramming" arguments that had developed in the humanities and law (e.g., Wigmore Diagrams [97]) and it was from these developments that the current notion of a structured safety case and its attendant notations and tools emerged.

The key elements of a structured argument are a clear separation between the *high level argument* and its supporting *evidence* [131], and the expectation that the high level argument should be structured in a hierarchical fashion around "claims" (or "goals") each of which is justified by a "warrant" (or "strategy") based on evidence or lower-level claims [9]. The structure of the argument can be displayed visually (see, for example, Figures 4.2 or 4.4 on Pages 58 and 64, respectively) as a graph in which claims and evidence are represented as nodes and each claim has arcs connecting it via its warrant to its supporting claims and evidence (the graph will be a tree if each interior claim or item of evidence is used in support of only a single parent claim).

Some of those advocating structured arguments focussed on diagrammatic presentations (e.g., the Goal Structuring Notation, GSN [76,131], which is described in Section 4.2), while others focussed more on its content and the quality of justifications (e.g., Claims, Argument, Evidence, CAE [2], that is described in Section 4.1), but these are merely different paths to the same goal, which is to frame assurance for safety in terms of an argument, based on credible evidence, whose structure is clear and convincing. Chapter 3 of this report describes and illustrates topics in the construction of such arguments.

The developers of early structured safety cases were aware that the argument could not amount to a "proof" of safety: there are inherent uncertainties and incompletenesses in the evidence that can be collected (e.g., it is impossible to test every path in software of even moderate size), and in our knowledge of the system and its environment. Hence, they looked to the idea of an "inductive" argument to supply a framework for interpreting their cases. The term *inductive* is used with many senses in mathematics and logic: *mathematical induction* refers to a method for proving properties over the natural numbers (or, when generalized as *Noetherian Induction*, over any well-founded set), while induction in science generally refers to the "process" of inferring a general rule from limited observations. The sense, derived from informal logic, in which it is used here is "reasoning in which the premises seek to supply strong evidence for (not absolute proof of) the truth of the conclusion" [Wikipedia] (this is in contrast to a *deductive* argument, where the premises are intended to prove the conclusion). Unfortunately, there is no generally accepted method for assessing the "degree of soundness" (the standard term is *cogency*) of an inductive argument and so evaluation of safety case arguments is very much a matter of judgement [66]. Safety standards may stipulate that a case should be "compelling, comprehensible and valid" [124] but they lack strong guidance on how these should be assessed. Chapter 5 of this report examines these topics and, in particular, explores ideas for providing rigorous underpinnings for the evaluation of safety case arguments.

The main conclusion we wish to draw from this brief historical review is that safety cases are merely the most recent step in a steady evolution of methods for safety assurance that has been driven by increasing complexity and variety in the systems to be assured. Early methods focussed on specific hazards and stipulated particular methods of mitigation and development; later, as systems became more varied and complex, these topics were delegated to developers, who were now required to produce specified evidence to attest that their responsibilities had been performed adequately; in a safety case, the delegation is further extended to include selection of the evidence, and this must be justified by provision of an explicit argument.

The latest step in this evolution is from safety cases to *assurance cases*: these simply generalize the idea of assurance based on a structured argument from claims focussed on safety to those concerned with other important system properties, such as security, harm to the environment, or general dependability. Structured safety or assurance cases are now the required or recommended method of assurance in many fields. For example, the US Food and Drug Administration requires submissions for infusion pumps under the "510(k) Premarket Notification" procedure to provide a safety case [87]. The UK Health Foundation report [57] provides a useful review of the use of safety cases in six safety-critical industries: commercial aviation, automotive, defense, petrochemical, railways, and medical devices; details of each review are available online as supplements to the report.

We now proceed to consider assurance case principles, their notations and tools, and their evaluation. We are concerned primarily with the presentation and evaluation of cases, and not with the safety engineering that underlies their construction. A useful NASA reference that covers the latter material and integrates it with the idea of a "Risk-Informed Safety Case" is [33].

Chapter 3

Assurance Case Principles

We have reviewed some of the history leading to the introduction of structured assurance cases and we now turn to the content and presentation of such cases. The key idea is that a structured assurance case is composed of three elements: a *claim* that states the property to be assured, *evidence* about the design and construction of the system, and an *argument* that the evidence is sufficient to establish the claim.[1] It is not enough that all three elements are present: in combination, they must provide a “compelling, comprehensible and valid case” that a system is safe (or, more generally, satisfies the property stated in its claim) “for a given application in a given operating environment” [124].

This framework of claims, argument, and evidence is surely the (perhaps tacit) intellectual foundation of any rational means for assuring and certifying the safety or other critical property of any kind of system. However, assurance cases differ from other means of assurance, such as those based on standards or guidelines, by making all three components explicit. Standards and guidelines generally specify only the evidence to be produced; the claims and the argument are unstated, although an implied argument presumably informed the deliberations of the standards body concerned, and the claims are often derived from governing legislation or regulation.

A second way in which assurance cases differ from most other ways of assuring systems is in supporting—indeed requiring—a layered, hierarchical approach. This is the *structure* in structured assurance cases and it arises because an argument is not simply a monolithic narrative from evidence to claims but a series of argument *steps* that are chained together: each step uses evidence and lower-level claims to directly justify its local claim, which is then used by higher-level steps, so that the overall argument has a tree-like structure (see, for example, Figures 4.2 and 4.4). Of course, the evidence specified in a standards-based approach also can be

[1] This terminology has become standard in the field of assurance, but what is here called an *argument* was traditionally called a *warrant* and the term *argument* then referred to the whole “case” (see, for example, Toulmin [122]).

grouped hierarchically according to the artifacts it concerns (high-level vs. low-level requirements, for example), but the grouping is seldom clean-cut (for example, as we will see in Figure 3.1 $\boxed{\text{c}}$, a required item of evidence may concern hardware performance as well as high-level requirements) and a cleanly layered relationship between the groups is hard to establish in the absence of an explicit argument.

We will need to examine the hierarchical layering in structured assurance cases and must also inquire into the nature of claims and evidence, but first we should explore the content and interpretation of the individual steps of argument within a case. We will do this by building an assurance subcase around some of the evidence specified in the guidelines for certification of airborne software, DO-178C [99]; in particular, we will use the evidence specified for "Reviews and Analyses of High-Level Requirements" [99, Section 6.3.1 and Table A-3].

3.1 A Software Assurance Case

For those not familiar with assurance for aircraft software, the next three paragraphs provide a brief review the overall approach. More detailed, but still brief, overviews are available [6, 86, 109].

The upper levels of airplane development iterate through several levels of system design and safety analysis and some of these yield *System Requirements* (SR) for functions to be implemented in software. The processes leading to definition of the SR follow guidelines such as Aerospace Recommended Practice (ARP) 4761 [114] and 4754A [113] to provide assurance that higher-level hazards have been addressed and that each function will perform correctly and safely if its SR are satisfied. Assurance for the software that implements a function is accomplished by showing that it is a *correct* implementation of the SR, and this is the focus of the DO-178C guidelines. DO-178C describes 71 assurance "objectives," that may be required for airborne software. Not all 71 objectives are required for all software: the more serious the "failure condition" that could be precipitated by given software, the more assurance is required. We will return to this topic of "graduated assurance" and the related notion of "Software Level" at a later stage (see Section 5.2.3) but, for the time being, we will assume that all the objectives associated with a given activity must be satisfied (as is the case for "Level A" software).

Most standards-based approaches to software assurance expect software development to follow a systematic process that refines the software design through several levels of intermediate specifications and eventually to executable object code (EOC). Some guidelines are relaxed about the actual development process and require only that appropriate intermediate artifacts are produced (possibly as after-the-fact reconstructions, in the style of Parnas [89]), but the FAA conducts reviews at several "stages of involvement" during software development [39], so the major planning, development, and assurance steps do need to be structured and performed in a manner that is consistent with the expectations of DO-178C.

In DO-178C, the topmost level of software specification constitutes the *High-Level Requirements* (HLR), and the claim we wish to make about these is that they correctly represent the system requirements. This claim will mesh with an upper-level assurance case that the SR ensure safety, and a lower-level case that the software correctly implements the HLR, thereby providing a complete assurance case for safety of the software-implemented function.

To introduce issues that arise in construction of an assurance case, we will now examine argument steps for the claim that the HLR correctly represents the SR. DO-178C addresses this topic in its Section 6.3.1 and identifies seven objectives that can be required to discharge this (implicit) claim. DO-178C objectives are not the same as evidence and do not (in most cases) mandate specific forms of evidence; instead, they specify what the proffered evidence should accomplish. In the language of assurance cases, therefore, the objectives of DO-178C correspond to lower level claims or, as we will call them, subclaims. The relevant subclaims of DO-178C are shown in Figure 3.1; we take the "titles" of the subclaims from [99, Table A-3] and their interpretation (i.e, their `MEANS` clauses) is paraphrased from [99, Section 6.3.1]. DO-178C does not provide an argument to motivate and justify the selection of these specific subclaims, so our task is to reconstruct a plausible case. What we will do is first develop a simple argument that, in retrospect, we will find unsatisfactory. The purpose of this first argument is pedagogical: it provides a simple context in which to introduce several relevant topics. We will then construct a second, more refined argument, and then a third.

We begin by introducing the logical form of an individual argument step. Just like the full argument of a complete assurance case, individual steps in an argument establish claims based on evidence; the (sub)claim delivered by one step can be used as evidence by another, thereby linking the steps together to create the full argument.

An individual argument step is a little like an inference rule in logic, where truth of the premises establish truth of its conclusion. A logical rule may involve quite complex logical relations among its premises, as in the following example.

$$p_1 \texttt{ AND } (p_2 \texttt{ OR } p_3) \texttt{ AND } (\texttt{NOT } p_4) \texttt{ IMPLIES } c$$

In classical logic, the disjunction (p_2 `OR` p_3) means that one or the other of p_2 and p_3 is true, but we may not know which one (it is also possible that both are true). In an assurance case argument, however, we expect to establish the truth of each item of evidence, so disjunctions are inappropriate. Similarly, the logical negation `NOT` p_4 means that p_4 is false and this also is inappropriate when p_4 is interpreted as

SUBCLAIM

$\boxed{\text{a}}$ HLR comply with SR

MEANS

Ensure that the system functions to be performed by the software are defined, and that the functional, performance, and safety-related requirements of the system are satisfied by the HLR,. . . [2]

SUBCLAIM

$\boxed{\text{b}}$ HLR are accurate and consistent

MEANS

Ensure that each HLR is accurate, unambiguous, and sufficiently detailed, and that the requirements do not conflict with each other.

SUBCLAIM

$\boxed{\text{c}}$ HLR are compatible with target computer

MEANS

Ensure that no conflicts exist between the HLR and the hardware/software features of the target computer, especially system response times and input/output hardware.

SUBCLAIM

$\boxed{\text{d}}$ HLR are verifiable

MEANS

Ensure that each HLR can be verified.

SUBCLAIM

$\boxed{\text{e}}$ HLR conform to standards

MEANS

Ensure that the Software Requirements Standards were followed during the software requirements process and that deviations from the standards are justified.

SUBCLAIM

$\boxed{\text{f}}$ HLR are traceable to SR

MEANS

Ensure the functional, performance, and safety-related requirements of the system that are allocated to software were developed into the HLR.

SUBCLAIM

$\boxed{\text{g}}$ Algorithms are accurate

MEANS

Ensure the accuracy and behavior of the proposed algorithms, especially in the area of discontinuities.

Figure 3.1: Objectives/Subclaims from Section 6.3.1 and Table A-3 of DO-178C

[2] The clause concerning “Derived Requirements” is omitted here and will be described later.

evidence.[3] We conclude that the steps in an assurance case argument always have the logical form

$$p_1 \texttt{ AND } p_2 \texttt{ AND } \cdots \texttt{ AND } p_n \texttt{ IMPLIES } c. \tag{3.1}$$

In logic, this form is called a *Horn Clause* and it has some properties that may be useful when we come to examine the evaluation of assurance cases.[4] Now an argument is not (just) logic, and to develop more of its attributes we need to shift from the austere notation of logic to something that allows us to say more about the rôles played by its parts. There are several such notations, mostly graphical, but before describing them we introduce the concepts using a simple textual presentation. In this form, the basic argument step (3.1) can be presented as follows.

```
ARGUMENT name
CLAIM
 claim
PREMISES
 premise_1, premise_2, ..., premise_n
END ARGUMENT
```

The interpretation here is that if all the premises are true, then truth of the claim must follow. Implicitly, this a strong or, as we will say, a *deductive* argument; a weaker kind of argument is an *inductive* one in which truth of the premises strongly suggests, but does not guarantee, truth of the claim. Classical formal logic deals only with deductive arguments, and exploring suitable treatments for inductive arguments will occupy much of Chapter 5. Below, we will add the keywords `DEDUCTIVE` or `INDUCTIVE` to make explicit the intended interpretation of each `ARGUMENT`.

We now turn to formulation of DO-178C Section 6.3.1 as an assurance case argument; as noted earlier, on Page 27, the claim this section of DO-178C seeks to justify is that the "`HLR correctly represent the SR`." Our first step is to elaborate what this should mean. Useful clues are given in [99, Section 6.1], which concerns the "purpose of software verification." This section says the purpose is detection and reporting of errors that may have been introduced during software development, and it lists six attributes that should be verified. Of these, the only one that concerns both SR and HLR is the following.

6.1.a. The SR allocated to software have been developed into HLR that satisfy those SR.

[3] Evidence may establish a negative property such as "this program will generate no runtime exceptions," but this is not the same as generating negative evidence for the positive claim "this software may generate runtime exceptions."

[4] The implication is equivalent to $\texttt{NOT } p_1 \texttt{ OR NOT } p_2 \texttt{ OR } \cdots \texttt{ OR NOT } p_n \texttt{ OR } c$: i.e., a disjunction with (at most) one positive term, which is the definition of a Horn Clause [64]; Horn clauses will be familiar to those who know logic programming languages such as Prolog.

Another refers to "software requirements" without specifying whether these are SR, HLR, or lower-level requirements:

> **6.1.d.** The Executable Object Code satisfies the software requirements (that is, intended function), and provides confidence in the absence of unintended functionality.

Drawing on these, we can identify two properties that constitute the top-level claim: everything in the SR is correctly represented in the HLR, and there is nothing in the HLR that does not come from the SR. We can document this as follows.

```
                                                                    1
CLAIM
  HLR correctly represent SR
MEANS
     (i) All the functionality and properties required by the SR are specified
     correctly in the HLR and (ii) there is no functionality specified in the
     HLR that is not required by the SR.
END
```

When we look at the subclaims offered by DO-178C in Figure 3.1, we see that the first $\boxed{a}$ "`HLR comply with SR`" closely matches (i) above, and that the sixth $\boxed{f}$ "`HLR are traceable to SR`" could be interpreted to discharge (ii). Hence our first argument step looks as follows. At the moment, we do not know whether the subclaims are sufficient to ensure the claim, so we mark the argument as inductive.

```
INDUCTIVE ARGUMENT DO-178C HLR Verification Process                 2
CLAIM
 HLR correctly represent SR
PREMISES
 HLR comply with SR,
 HLR are traceable to SR
END ARGUMENT
```

Each argument step is expected to be supported by some reason or justification why the premises are considered to establish the claim. Here, it is the observation that the two subclaims chosen from DO-178C closely match the two parts of the claim. We can record this as follows.

```
JUSTIFICATION                                                       3
  HLR verification rationale
MEANS
     The claim in this argument has two parts: everything in the SR is
     correctly specified in the HLR, and everything in the HLR is required
     by the SR. The two items of required evidence exactly correspond to
     these two parts.
END
```

Then we link this into the argument by extending 2 with the text marked in blue.

```
INDUCTIVE ARGUMENT DO-178C HLR Verification Process                    4
CLAIM
 HLR correctly represent SR
PREMISES
 HLR comply with SR,
 HLR are traceable to SR
JUSTIFICATION
 HLR verification rationale
END ARGUMENT
```

For this justification to be sound, we need to be sure that the text that box 3 cites from box 1 and from a and f of Figure 3.1 really do bear the interpretations we are placing on them. Here, we are very reliant on rather terse natural language descriptions that may be imprecise or even ambiguous. For example, part (ii) of 1 requires there is nothing in the HLR that does not come from (i.e., is not traceable to) the SR, and the justification 3 states that f provides the evidence to ensure this. Now, the title of f, "HLR are traceable to SR," supports this interpretation, but its body suggests that tracing goes the other way: to ensure that the "requirements of the system" (i.e., SR) "were developed into the HR". [5]

This kind of difficulty is be expected: DO-178C was not developed with an explicit argument, and the argument fragment that we are developing here is for the purpose of illustrating issues and concepts rather than retrospectively constructing a best justification for Section 6.3.1 of DO-178C. It is not my purpose to criticize DO-178C, but notice that the mere act of trying to frame an argument around these topics brought this issue to our attention in a manner that seems reliable, and this is surely a valuable benefit.

In a real assurance case we would need to resolve the issue concerning traceability by adjusting either the argument or the meanings attached to its subclaims, but here we will leave things as they are and press on to related topics. The first of these is that even when we are satisfied with an argument, we may still have concern that it amounts to less than definitive proof. This may be because we have doubts about the strength of the evidence that will be used to discharge some of the subclaims (e.g., software testing is seldom exhaustive, so it can provide good but not perfect evidence), or about the validity of an argument step itself. Consequently, we may wish to require some additional evidence that does not add to the logical inference that the claim follows from the cited evidence or subclaims, but that increases our *confidence* that it does do, or that supplied evidence will truly discharge the subclaims. (The rough idea is that confidence claims, if true, bolster our belief in

[5] Readers who find it difficult or tedious to follow all the "box" cross-references in this paragraph will surely agree that tool support for assurance cases must include some form of hypertext browsing.

the other subclaims or in the argument but, unlike conventional subclaims, their falsity does not invalidate the claim.)

In the current case, it might be that we will have more confidence in the argument if the HLR satisfies some "quality" measure: this would give us more confidence that whatever evidence is supplied to satisfy the subclaim "`HLR comply with SR`" has at least been working from a "high quality" HLR. Looking at the repertoire of objectives in Figure 3.1, we see that $\boxed{b}$ "`HLR are accurate and consistent`," $\boxed{d}$ "`HLR are verifiable`," and $\boxed{g}$ "`Algorithms are accurate`" seem suitable.[6]

We can then add these items to our argument, but in a subclause that indicates they are not part of the basic argument, but are there to add to our confidence in it.

```
INDUCTIVE ARGUMENT DO-178C HLR Verification Process                   5
CLAIM
 HLR correctly represent SR
PREMISES
 HLR comply with SR,
 HLR are traceable to SR
JUSTIFICATION
 HLR verification rationale
CONFIDENCE
 HLR are accurate and consistent,
 HLR are verifiable,
 Algorithms are accurate
END ARGUMENT
```

We might also feel more confident if the process of developing the HLR conformed to suitable standards, as described in Figure 3.1 $\boxed{e}$ "`HLR conform to standards`." This could be seen as a further contribution to be added to the confidence section of the argument, but we might alternatively consider it a sufficiently different strand that it deserves its own heading. We will consider the interpretation to be attached to confidence and standards claims in 5.2.1, but my opinion is that all these claims have the same logical standing so the keywords `CONFIDENCE` and `STANDARDS` can be treated as synonyms (or, equivalently, `STANDARDS` is just a comment labeling some of the `CONFIDENCE` claims).

[6] I stress that in a real assurance case the argument, subclaims, and evidence should be developed in an integrated manner, not selected "off the shelf"; I am doing so here simply to illustrate the components of arguments.

```
INDUCTIVE ARGUMENT DO-178C HLR Verification Process                6
CLAIM
 HLR correctly represent SR
PREMISES
 HLR comply with SR,
 HLR are traceable to SR
JUSTIFICATION
 HLR verification rationale
CONFIDENCE
 HLR are accurate and consistent,
 HLR are verifiable,
 Algorithms are accurate
STANDARDS
 HLR conform to standards
END ARGUMENT
```

We have now used six of the seven objectives of Figure 3.1 as subclaims in our argument; this argument concerns the relationship between the SR and HLR and it is therefore to be expected that its subclaims concern one or the other, or both, of these artifacts. The remaining objective of Figure 3.1, $\boxed{c}$ "`HLR are compatible with target computer`," is rather different in that it refers to an "outside" artifact, namely the "target computer."

The motivation here seems to be one of prudent workflow: the SR and HLR certainly need to be cognizant of the computational capabilities of the platform that will execute the developed program, but the details of these capabilities surely are more properly the focus of later stages of software development, such as architecture definition, low-level requirements, and coding. On the other hand, it is possible that one way of formulating the HLR will render them incompatible with the target computer, while another does not suffer from this defect.

An example may help make this clear. About 20 years ago, I was involved in a project to explore application of formal methods to requirements evaluation of software change requests for the flight software of the Space Shuttle [51]. The function we worked on was called "Jet Select" and its purpose was to select the three reaction control jets that should be fired to accomplish a desired rotational maneuver, specified as a vector. The existing algorithm, called "max dot product," picked the three jets whose individual thrust vectors had the largest scalar (i.e., "dot") products with the desired vector. A disadvantage of this algorithm was that it sometimes picked jets that almost opposed each other, thereby wasting fuel and reducing performance. The proposed new algorithm was called "min angle" and the idea was to pick the three jets whose resultant was closest (measured by angle) to the desired vector. The SR were reverse-engineered from the simulation code that had been used to validate this algorithm and featured a triply nested loop. Many months into the requirements review and HLR development process, it was discovered that there was no way to execute a triply nested loop in the few milliseconds available on the ancient computers of the Space Shuttle. And because the SR specified a triply-nested loop, the HLR could not depart from this. If the SR had instead specified some satisficing constraint, it might have been possible to develop an effective implementation.

So, it seems reasonable that this objective should be added to the evolving argument as an additional confidence measure that we label `WORKFLOW`, as shown below. (It is plausible that "`Algorithms are accurate`" could go under this label, too.)

```
INDUCTIVE ARGUMENT DO-178C HLR Verification Process                      7
CLAIM
 HLR correctly represent SR
PREMISES
 HLR comply with SR,
 HLR are traceable to SR
JUSTIFICATION
 HLR verification rationale
CONFIDENCE
 HLR are accurate and consistent,
 HLR are verifiable,
 Algorithms are accurate
STANDARDS
 HLR conform to standards
WORKFLOW
 HLR are compatible with target computer
END ARGUMENT
```

However, as we noted earlier, whereas the other subclaims in the argument concern the SR and the HLR, this one also concerns the target computer. It seems sensible to annotate each argument step with the artifacts that it references, so that we will know which steps need to be reexamined when artifacts are changed. We can call this annotation the "context" of the argument step and add it to the argument as follows.

8

```
CONTEXT toplevel
USES
 SR-docs, HLR-docs, target-computer-docs

INDUCTIVE ARGUMENT DO-178C HLR Verification Process
CLAIM
 HLR correctly represent SR
PREMISES
 HLR comply with SR,
 HLR are traceable to SR
JUSTIFICATION
 HLR verification rationale
CONFIDENCE
 HLR are accurate and consistent,
 HLR are verifiable,
 Algorithms are accurate
STANDARDS
 HLR conform to standards
WORKFLOW
 HLR are compatible with target computer
END ARGUMENT
```

The idea here is that the argument step we are developing is part of a context called `toplevel` and it references documents associated with the SR, the HLR, and the target computer.

A problem with adding the target computer to our context is that it is a large artifact that is likely to undergo changes long after construction of this argument step; we must always consider the consequences of any change (and, indeed, an assurance case should surely be bound under configuration control to the artifacts concerned) and will therefore need to reexamine this argument step each time a change is made to the target computer.

Coping with changes in "distant" artifacts is just one manifestation of a fundamental issue in assurance cases: there must be some "modular" way to construct and evaluate a case, otherwise it will exceed our intellectual grasp and our ability to manage it. The feasibility and soundness of modular (or "compositional") reasoning is a function of the system's design (e.g., whether it has strong barriers to propagation of misbehavior), how it is developed (e.g., whether knowledge of one component's internal design can influence development of another), how its assurance evidence is gathered (e.g., whether components are tested in environments where other components are represented by their specification, not their implementation), and (our focus here) how its assurance case is structured and presented.

Here, we need some way to reference attributes of the target computer that are relevant to the local argument without having to add everything about the target computer to our context. A natural way to do this is by referencing a suitable claim about that artifact. Thus, we might require that elsewhere in the full assurance case, a subcase concerned with properties of the target computer should establish a claim like the following (with suitable values substituted for the parameters a, r, d, and s).

9

```
CONTEXT lowlevel
USES target-computer-docs

CLAIM
  Target Hardware Performance
MEANS
     Tasks of type a are deterministically scheduled every r msec. for a
     duration of d msec. on a processor that averages s instructions per
     second.
END
```

Outside of the `lowlevel` context, we can reference this claim as `lowlevel.Target Hardware Performance` and can then import it into our local argument as an assumption, as follows. Notice that we have now dropped `target-hardware-docs` from this context.

10

```
CONTEXT toplevel
USES
 SR-docs, HLR-docs

INDUCTIVE ARGUMENT DO-178C HLR Verification Process
ASSUMPTIONS
 lowlevel.Target Hardware Performance
CLAIM
 HLR correctly represent SR
PREMISES
 HLR comply with SR,
 HLR are traceable to SR
JUSTIFICATION
 HLR verification rationale
CONFIDENCE
 HLR are accurate and consistent,
 HLR are verifiable,
 Algorithms are accurate
STANDARDS
 HLR conform to standards
WORKFLOW
 HLR are compatible with target computer
END ARGUMENT

...

END CONTEXT
```

Any evidence or subargument used to validate the subclaim `HLR are compatible with target computer` can reference the parameters a, r, d, and s and will need to be reconsidered only if the status of the subclaim `lowlevel.Target Hardware Performance` should change.

In this example, the nonlocal claim was used as an assumption, but it also seems reasonable that nonlocal claims can be referenced as general subclaims in the argument or confidence section (though obviously not as the claim being established). Such nonlocal references (and, indeed, local references that are not strictly hierarchical) mean that an assurance case argument may not be a tree but rather a graph (having the form of a "tree with cross-links"). It will be necessary to ensure that such nonhierarchical references do not introduce circularity.

We rather casually stated that "`lowlevel.Target Hardware Performance` can be used as an assumption, so we should next examine the nature of assumptions in a little more detail.

In logic, assumptions and side conditions are used to restrict application of rules or axioms. For example, we might have

$$x \times \frac{y}{x} = y, \;\; \texttt{ASSUMING} \;\; x \neq 0. \tag{3.2}$$

In general, assumptions can be interpreted as additional premises, so that (3.2) can be treated as

$$x \neq 0 \;\; \texttt{IMPLIES} \;\; x \times \frac{y}{x} = y.$$

Under an assumption a, our generic argument step (3.1) from page 29 becomes

$$p_1 \;\texttt{AND}\; p_2 \;\texttt{AND}\; \cdots \;\texttt{AND}\; p_n \;\texttt{IMPLIES}\; c, \;\; \texttt{ASSUMING}\; a$$

and this is therefore interpreted as

$$a \;\texttt{IMPLIES}\; (p_1 \;\texttt{AND}\; p_2 \;\texttt{AND}\; \cdots \;\texttt{AND}\; p_n \;\texttt{IMPLIES}\; c),$$

which simplifies under the laws of logic to

$$a \;\texttt{AND}\; p_1 \;\texttt{AND}\; p_2 \;\texttt{AND}\; \cdots \;\texttt{AND}\; p_n \;\texttt{IMPLIES}\; c.$$

Hence, the interpretation of $\boxed{10}$ is that the subclaim named as an assumption is conjoined to the other subclaims in the main part of the argument and to each of the subclaims in the confidence part. Of course, this particular assumption is unnecessary to all except the last of these, so a more sophisticated notation might allow assumptions to be attached to just the relevant parts of an argument rather than the whole thing.

Assumptions raise another topic that is worth attention. In some languages, a term such as $\frac{y}{x}$ does not "make sense" unless $x \neq 0$, so we should not even inspect the expression in (3.2) until we know its assumption is true. In general, since all the premises to an assurance case argument must be true if we are to conclude its claim, we could allow each premise to be interpreted only under the assumption that all the other premises and any explicit assumptions are true. However, it can require additional analysis to ensure there is no circularity in this reasoning, so a useful compromise is to impose a left-to-right reading. In concrete terms, this means that each subclaim or item of evidence named in an argument step such as $\boxed{10}$ is evaluated assuming the truth of all subclaims or evidence appearing earlier in the argument step.

Notice this treatment means there is nothing special about assumptions: any subclaim appearing early in the argument step will have the same impact. The textual notation we are using here allows its components to be divided into several parts and to appear in any order, and this can be used to exploit this left to right (or top to bottom) accumulation of assumptions as in the following reordering (which serves no purpose other than to illustrate the idea), where we do retain the explicit labeling of the assumption to alert the human reader of the intuitive purpose of this subclaim.

```
INDUCTIVE ARGUMENT DO-178C HLR Verification Process rearranged      11
ASSUMPTIONS
 lowlevel.Target Hardware Performance
CONFIDENCE
 HLR are accurate and consistent,
 HLR are verifiable
 Algorithms are accurate
PREMISES
 HLR comply with SR
WORKFLOW
 HLR are compatible with target computer
STANDARDS
 HLR conform to standards
CLAIM
 HLR correctly represent SR
PREMISES
 HLR are traceable to SR
JUSTIFICATION
 HLR verification rationale
END ARGUMENT
```

The idea here is that subclaims labeled PREMISES or ASSUMPTIONS are treated as premises, subclaims labeled CONFIDENCE, STANDARDS or WORKFLOW are treated as confidence items, and anything labeled CLAIM is treated as the claim or goal; all subclaims are interpreted in left to right/top to bottom order, except the CLAIM, which is always interpreted last.

We have completed construction for an argument step that reflects Section 6.3.1 of DO-178C and now need to consider how its subclaims will be discharged. We can take $\boxed{g}$ Algorithms are Accurate as an example and suppose that the evidence used to discharge this claim is a combination of expert analysis and simulation.

```
INDUCTIVE ARGUMENT Algorithm aspects                                12
CLAIM
 Algorithms are Accurate
EVIDENCE
 Expert review and simulation
JUSTIFICATION
 Explanation of expert review and simulation
END ARGUMENT
```

Elsewhere, we will provide (presumably lengthy) descriptions of the meanings attached to the EVIDENCE and the JUSTIFICATION cited in this argument step. In interpreting an argument step, EVIDENCE is treated just like PREMISES but the keyword indicates that this thread of the argument terminates at this point. (Dually, we could use the keyword SUBCLAIM to introduce premises that will themselves be the subject of other argument steps.)

It is plausible that an argument step might include both EVIDENCE and PREMISES as illustrated by the top claim C on the left of Figure 3.2, but I think there is value in restricting arguments to what I will call "simple form" in which one or the other may appear, but not both (later, I will add a small qualification to this restriction). Any argument can be converted to simple form by introducing additional subclaims as shown on the right of Figure 3.2. I will call an argument step that has only subclaims (in diagram form, a box with only boxes below it) an *interior* or *reasoning* step, and one that has only evidence (in diagram form, a box with only circles below it) an *evidential* step.

Thus, in Figure 3.2, the non-simple argument step $\mathbf{AS}_1$ on the left is converted into the simple reasoning step $\mathbf{RS}_1$ on the right by introducing the new evidential step $\mathbf{ES}_\mathrm{N}$ with subclaim $\mathbf{SC}_\mathrm{N}$; the already simple argument step $\mathbf{AS}_2$ is relabeled as an evidential step $\mathbf{ES}_2$.

My reason for preferring simple form is that the type of justification is likely to be different for the two kinds of argument step. Evidence is about our knowledge and understanding of the actual artifacts that comprise the design and its environment: it is our *epistemology*. Premises, on the other hand, reflect our reasoning about the system: they are our *logic*. To be confident in an assurance case, we need to be confident in our reasoning, given our knowledge, *and* confident in our knowledge: that is, we need to be confident in both our logic and our epistemology.

The justification provided for an evidential step must explain why and how this evidence about the real world supports the claim: it is about the bridge from epistemology to logic. When multiple items of evidence are cited in support of a claim, they are generally interpreted *cumulatively*: each adds some "weight" to our confidence in the claim. The justification for an interior step, however, is about reasoning: it has to explain why the claim logically follows from the subclaims. When multiple subclaims are used to support a claim, they are generally interpreted as a *conjunction*: all of them are needed to compel confidence in the claim. Later, I will argue that while evidential steps are necessarily inductive, interior steps should be deductive.

In DO-178C, the evidence required for the subclaims of Section 6.3.1 is described in its Table A-3. This specifies that for Level A software (i.e., software whose malfunction could lead to catastrophic failure conditions) the evidence must be constructed "with independence" meaning that it should be performed by a different team than that responsible for developing the software. This seems like a confi-

Here, **C** indicates a claim, **SC** a subclaim, and **E** evidence; **AS** indicates a generic argument step, **RS** a reasoning step, and **ES** an evidential step.

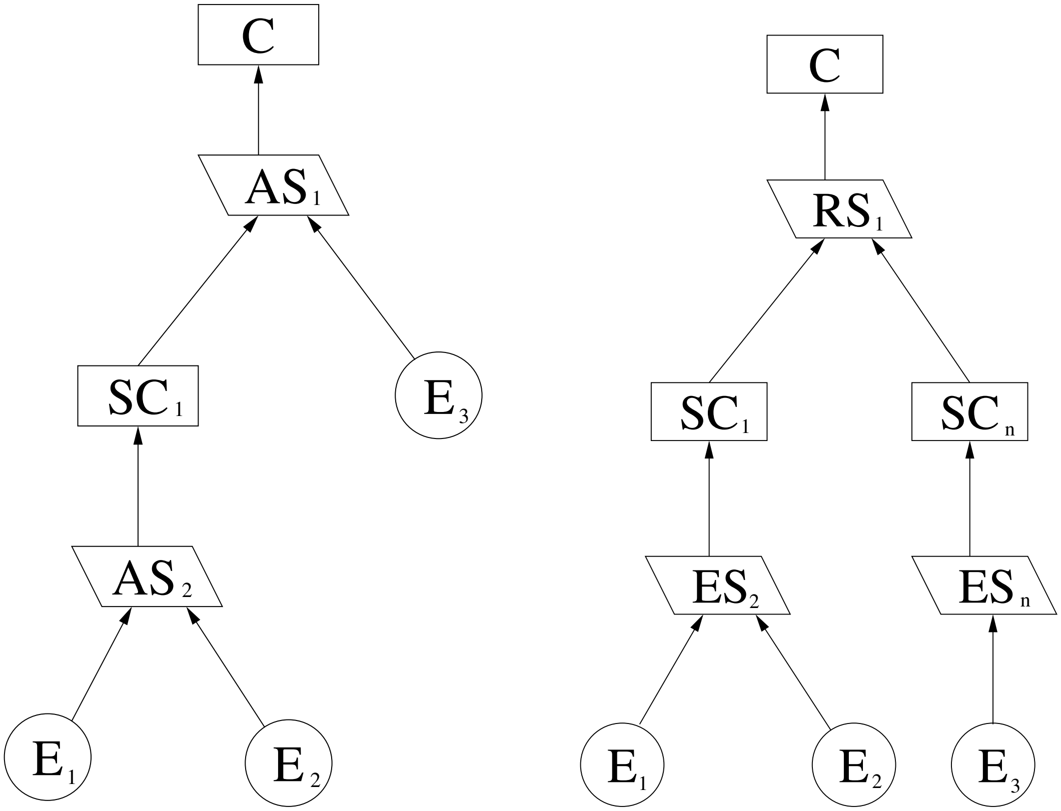


Figure 3.2: Converting an Argument from Free (left) to Simple Form (right)

dence measure, so we add it (and, elsewhere, its meaning) to produce the following modification of 12, where we also change the annotation to indicate that this is an evidential argument step.

```
EVIDENTIAL ARGUMENT Algorithm aspects Level A                         13
CLAIM
 Algorithms are Accurate
EVIDENCE
 Expert review and simulation
CONFIDENCE
 Independence
JUSTIFICATION
 Explanation of expert review and simulation
END ARGUMENT
```

Table A-3 of DO-178C also specifies that for Level D software (i.e., that whose malfunction can lead to only "minor" failure conditions) this subclaim is not required at all. There seem to be two ways to deal with "graduated assurance" of this kind (see Section 5.2.3): one is to have a (possibly) different argument for each level (so the Level D version of [11] would omit this subclaim); the other is to retain the same argument but supply different evidence. We could accommodate the latter by introducing the keyword NONE to indicate that a claim is not supported.

```
EVIDENTIAL ARGUMENT Algorithm aspects Level D                          14
CLAIM
 Algorithms are Accurate
EVIDENCE
 NONE
JUSTIFICATION
 This claim is not required for Level D
END ARGUMENT
```

We have now seen all the main elements of arguments in assurance cases: the CLAIM, and its JUSTIFICATION based on PREMISES, which may be subclaims, evidence, or assumptions, supported by additional subclaims that do not contribute to the logical part of the argument but increase our CONFIDENCE about it in various ways. We also saw the importance of narrowly defined CONTEXT and the value of referencing claims from other contexts in enabling the construction of arguments that are somewhat modular or compositional,

The assurance case that we formed around Section 6.3.1 of DO-178C served as a vehicle to introduce and illustrate these concepts but now that we have done that, it is appropriate to pause and reconsider that case. The main argument in [10] (or [11]) has two premises and five additional subclaims used for confidence and we can ask if this is a suitable balance. A full assurance case consists of a cross-linked tree of argument steps with evidence at the leaves. Each argument step provides a justification why the truth of its premises is sufficient to conclude the truth of its claim, when supported in some way by the confidence claims. As we explained above, in an argument of simple form, the justifications at the leaves are about our epistemology, while those in the interior are about our reasoning or logic. It follows that any doubts that might reduce our belief in the case are also of two kinds: we can have "logic doubt" about our reasoning, or "epistemic doubt" about our evidence. So how can confidence claims help either of these?

My belief is that logic doubts should be eliminated if at all possible: we are free to choose our argument and its subclaims and, if we harbor doubt about these, we should revise them. Thus, I think it is best if the interior or reasoning steps of an argument are deductive; by its very nature, the premises to a deductive step are sufficient to compel belief in its conclusion; hence, confidence claims add nothing to a deductive argument and should not be used.

If a system or its assurance case are so complex that it is considered impossible to formulate all its reasoning steps in deductive form, then it may be acceptable to employ inductive arguments supported by confidence claims. We will see, in Section 5.2.1, that there seems no rigorous way to evaluate an inductive reasoning step (with or without confidence claims), so there is strong dependence on good judgment. Suppose for example, that in [2] we had used the subclaim $\boxed{\mathrm{g}}$ "`Algorithms are accurate`" in place of $\boxed{\mathrm{f}}$ "`HLR are traceable to SR`" as shown below.

```
INDUCTIVE ARGUMENT Flawed DO-178C HLR Verification Process
CLAIM
 HLR correctly represent SR
PREMISES
 HLR comply with SR,
 Algorithms are accurate
CONFIDENCE
 HLR are traceable to SR
END ARGUMENT
```

This alternative argument is not logically persuasive, and no addition of confidence items can make it so; even the addition of $\boxed{\mathrm{f}}$ "`HLR are traceable to SR`" does not repair the argument if this is merely a confidence item.

However, confidence claims can be appropriate for the leaf or evidential steps of an argument, where they can provide additional information about our epistemology and increase our belief in the inductive justification from evidence to claim. Even here, however, I am inclined to think that many confidence claims would be better used as direct evidence or assumptions. In [13], for example, we establish the claim "`Algorithms are accurate`" through analysis and simulation. The justification for this argument step should explain what we did (and why it is sufficient), and how we did it. Hence, the evidence must address these topics, and the claim of `Independence` speaks to "how we did it"—thus, it is a necessary item of evidence, not a mere confidence factor. However, as we will see in Sections 5.2.1 and 5.2.2, confidence is not strongly distinguished from evidence in the interpretation of these steps, and so the choice of label serves human comprehension rather than technical purposes.

These recommendations—that interior argument steps should be deductive, and that confidence claims should be avoided—are radical: in current practice, assurance cases are regarded as quintessentially "inductive" arguments; that is the truth of the premises should provide strong and convincing evidence for the truth of the conclusion, but is not expected to provide incontrovertible "proof." I accept this, but the question is: where do we allow doubts to reside—in the sufficiency of our evidence, in the quality of our reasoning, or both? We will return to this in Chapter 5 but the motivation for my recommendation is that there no good way to evaluate an argument in which we have logic doubt, and no good way to evaluate the contribution of confidence claims to reasoning steps. If we cannot credibly assess how much doubt we have, nor how much is acceptable, then allowing any doubt at all puts us on a slippery slope; hence, the recommendation that reasoning steps should be deductive and confidence claims eschewed. Inductive evidential steps do not pose the same difficulties because they are evaluated epsistemically rather than logically: we do not ask whether the conjunction of evidence implies the claim, but whether the combined "weight" of evidence is sufficent to persuade us of the claim. This epistemic approach cannot be extended to inductive reasoning steps without abrogating their rôle as reasoning steps: the local subargument becomes simply a collection of evidence.

Our recommendations may seem a counsel of perfection: fine in theory but unattainable in practice. We discuss their feasibility in the Conclusion chapter, but note here that software assurance cases, where the top claim is correctness, may lend themselves more readily to deductive arguments than other cases, where the top claim is a system property such as safety

Observe that requiring interior argument steps to be deductive has some impact on the use of nugatory evidence as in the Level D example [14]. When we evaluate the soundness of interior argument steps, we surely assume that all subclaims will eventually be validated, but this assumption is repudiated by nugatory evidence such as [14]. We will return to this topic in Section 5.2.3.

First, however, we revisit the case constructed for DO-178C Section 6.3.1 in light of the discussion above that casts doubt on the value of confidence claims.

One alternative approach would return to our first step [2] and propose that, in addition to [a] "`HLR comply with SR`" and [f] "`HLR are traceable to SR`," we should also require that the HLR are "`fit for purpose`" and then declare the argument deductive. What were formerly confidence items, then become the evidence to discharge this new subclaim, as follows.

15

```
DEDUCTIVE ARGUMENT DO-178C HLR Verification Process
CLAIM
 HLR correctly represent SR
PREMISES
 HLR comply with SR,
 HLR are traceable to SR,
 HLR is fit for purpose
JUSTIFICATION
 revised HLR verification rationale
END ARGUMENT

ARGUMENT DO178C HLR fit for purpose
CLAIM
 HLR is fit for purpose
ASSUMPTIONS
 lowlevel.Target Hardware Performance
PREMISES
 HLR are accurate and consistent,
 HLR are verifiable,
 Algorithms are accurate,
 HLR are compatible with target computer,
 HLR conform to standards,
JUSTIFICATION
 HLR fit for purpose rationale
END ARGUMENT

EVIDENTIAL ARGUMENT Algorithm aspects Level A
CLAIM
 Algorithms are Accurate
EVIDENCE
 Expert review and simulation,
 Independence
JUSTIFICATION
 Explanation of expert review and simulation and value of independence
END ARGUMENT
```

Another point of view is that the argument in support of the claim "`HLR correctly represent SR`" is deductively sound without the addition of "`fit for purpose`" and the rôle of the latter is not to bolster the argument, but to provide confidence for the *evidence* submitted to discharge its subclaims. For example, evidence in support of the subclaim "`HLR comply with SR`" is likely to be the result of human review and traceability checking. The credibility of this review is surely contingent on the quality of the HLR: for example, no conclusion can be trusted if the HLR are inconsistent. Hence, it is reasonable to use "`HLR is fit for purpose`" as a confidence item for the evidential step "`HLR Compliance`" as shown in 16.

```
DEDUCTIVE ARGUMENT DO-178C HLR Verification Process                   16
CLAIM
 HLR correctly represent SR
PREMISES
 HLR comply with SR,
 HLR are traceable to SR
JUSTIFICATION
 Simplified HLR verification rationale
END ARGUMENT

EVIDENTIAL ARGUMENT HLR Compliance
CLAIM
 HLR comply with SR
EVIDENCE
 Human review and tracing
CONFIDENCE
 HLR is fit for purpose
JUSTIFICATION
 Description of procedures for human review and tracing and their
 reliability, and the results obtained
END ARGUMENT
```

I have no strong opinion of which of these (and other) plausible formulations of DO-178C Section 6.3.1 as an assurance case is the "best" or is closest to the intent of the document authors. I do hope, however, that this section has provided those having special expertise in DO-178C with information that enables them to explore this topic themselves, and to ask if DO-178C might have been improved, or at least framed differently, if assurance case principles had been considered during its development.

3.1.1 Derived Requirements

The assurance case implicit in DO-178C is essentially a correctness argument: it is assumed that the SR ensure safety—if they are implemented correctly—and the responsibility of the software assurance case is to establish that correctness. It is the responsibility of another part of the full assurance case to ensure that the SR do ensure safety (for aircraft, this case is implicit in guidelines such as ARP 4761 and ARP 4754A). However, new circumstances may come to light during software development that could affect safety: for example, a condition may be discovered where the SR do not define a behavior, or it may be impossible to achieve specified behavior under some conditions. In all these cases, the software developers must submit the circumstances to the systems engineers who will determine if new (or adjusted) requirements should be added to the SR. Such amendments to the SR are called “derived requirements” (the name is peculiar, since their essence is that these requirements were *not* derived by the primary top-down process).

Hence, a full software assurance case is not just about correctness: it must include a subargument and supporting evidence that software development includes a reliable process for identifying and notifying of circumstances that could require introduction of derived requirements. We note this expansion in the content of a software assurance case, but do not develop it further in our example.

3.2 A System Assurance Case

An assurance case for software correctness will be only one part of the assurance case for a complete system and, as we have seen, it tends to have a rather special form (because it is mostly about correctness, not safety), so before moving on to other topics we will look at a different example that sketches the case for a complete system. This example is from the GSN Standard [48], and is also discussed by Holloway [60].

The top-level claim that this (imaginary and otherwise unspecified) system is acceptably safe to operate is based on two subclaims: (a) all its hazards have been eliminated or mitigated (i.e., reduced in frequency or severity) to a sufficient degree, and (b) its software has been developed to a Software Integrity Level appropriate for the hazards involved.[7] This top level of the argument can then be recorded as follows.

```
CONTEXT control-system-example                                      17
USES
 control-sys-def-docs, operating-context-docs

INDUCTIVE ARGUMENT system-safety
CLAIM
 Control system is acceptably safe to operate
PREMISES
 All hazards eliminated or adequately mitigated,
 Software developed to SIL appropriate for hazards involved
JUSTIFICATION
 System hazards and software hazards are the only safety hazards
 and development to an appropriate SIL mitigates software hazards
END ARGUMENT
```

The `JUSTIFICATION` provided above is a placeholder: in a real assurance case a much more detailed explanation would be expected; the same applies to all the `JUSTIFICATION` sections below.

[7] Software Integrity Levels, or SILs, mainly appear in guidelines derived from IEC 61508 [69] and are similar in concept to the Software Levels, or DALs, of DO-178B/C, but the ordering is reversed: SIL 4 is the highest and SIL 1 the lowest (cf. Levels A to D in DO-178C).

For an assurance case about software correctness, as in the example used in the first part of this chapter, a standard method of argument is that everything present at one level of development (e.g., SR) is completely and correctly represented in the next level (e.g., HLR), and there is nothing in the lower level that does not come from the upper level. For an assurance case about a system design, common methods of argument are to reason over the components of the system (the step above implicitly does this—dividing the system into software and the rest), or over the hazards to the system. In the lower-level step below, we will use the "argument over hazards" style; we consider these stereotypical argument styles in more detail in Section 4.1.1.

To argue over hazards, we need to identify every hazard and its severity, and then justify subclaims that the system either eliminates or sufficiently mitigates each hazard. Here, it is assumed that the identified hazards are named H1, H2, and H3.

```
DEDUCTIVE ARGUMENT system-hazards-controlled                           18
CLAIM
 All hazards eliminated or adequately mitigated
ASSUMPTION
 The only system hazards are H1, H2, and H3,
 The worst case severity of H2 is XX,
 The worst case severity of H3 is YY
PREMISES
 Hazard H1 has been eliminated,
 Probability of H2 is below 10E-6 per year,
 Probability of H3 is below 10E-3 per year
JUSTIFICATION
 Some discussion, presumably referencing regulations from the field
 concerned, why these probabilities are sufficient for the hazard
 severities concerned
END ARGUMENT
```

We have recorded the list of identified hazards and their severities as ASSUMPTIONS; these are logically no different than other premises (recall the discussion on page 38) but the label is useful to indicate that they establish the context for the other premises.

We have labeled this argument DEDUCTIVE, meaning that we believe the claim is guaranteed to be true if the premises are. This belief is flawed, since the argument step assumes the hazards are independent and does not consider *combinations* of hazards. One hopes that these oversights would be identified and rectified during review, and it is my opinion that by labeling the argument DEDUCTIVE we raise the obligations on such reviews. Since we are merely presenting an existing example here, we note this weakness and move on; however, we will return to it in Section 4.1.1.

Next, we provide evidence for each of the subclaims used in the step above.

19

```
EVIDENTIAL ARGUMENT all-hazards
CLAIM
 The only system hazards are H1, H2, and H3,
 The worst case severity of H2 is XX,
 The worst case severity of H3 is YY
EVIDENCE
 Description of hazard analysis performed
JUSTIFICATION
 Some discussion why the method of hazard analysis is believed to
 identify all hazards, and how the worst-case severities are derived
END ARGUMENT

EVIDENTIAL ARGUMENT H1-eliminated
CLAIM
 Hazard H1 has been eliminated,
EVIDENCE
 Formal verification
JUSTIFICATION
 Explanation why the formal verification is credible and how the
 formal text relates to reality
END ARGUMENT

EVIDENTIAL ARGUMENT H2-mitigated
CLAIM
 Probability of H2 is below 10E-6 per year
EVIDENCE
 Fault-tree analysis
JUSTIFICATION
 Explanation why the fault-tree analysis is credible
END ARGUMENT

EVIDENTIAL ARGUMENT H3-mitigated
CLAIM
 Probability of H3 is below 10E-3 per year
EVIDENCE
 Fault-tree analysis
JUSTIFICATION
 Explanation why the fault-tree analysis is credible
END ARGUMENT
```

In practice, it is likely that the steps above will each be replaced by, or will reference, standardized and approved subarguments for the efficacy of the methods employed for hazard analysis, formal verification, fault-tree analysis, and the architectural modeling required to support these.

Notice that three claims are established by the first of the argument steps above. We could equivalently have written a single conjoined claim—or we could have provided a separate argument step for each claim, all citing the same evidence and related justification.

We now turn to the second premise of 17, which concerns the safety of the software used in the system.

```
DEDUCTIVE ARGUMENT SW-SILs                                          20
CLAIM
 Software developed to SIL appropriate for hazards involved
ASSUMPTION
 All hazards due to software identified and their severity assigned
PREMISES
 Primary system SW developed to SIL 4,
 Secondary system SW developed to SIL 2
JUSTIFICATION
 Some argument about why SIL 4 and 2 are appropriate for the
 identified hazard severities
END ARGUMENT
```

Again, it is likely there will be standardized subarguments relating the adequacy of various SILs in mitigating hazards due to software. These subarguments would likely resemble those that could be developed for DO-178C.

Finally, we present the evidence that software hazards were identified correctly and the software was developed to the appropriate SILs.

21

```
EVIDENTIAL ARGUMENT SW-hazards
CLAIM
 All hazards due to software identified and their severity assigned
EVIDENCE
 Description of hazard analysis performed
JUSTIFICATION
 Some discussion why the method of software hazard analysis is
 believed to be effective
END ARGUMENT

EVIDENTIAL ARGUMENT Primary-SIL
CLAIM
 Primary system SW developed to SIL 4
EVIDENCE
 Evidence that Primary SW was developed to SIL 4
JUSTIFICATION
 Comparison of the evidence against the requirements for SIL 4
END ARGUMENT

EVIDENTIAL ARGUMENT Secondary-SIL
CLAIM
 Secondary system SW developed to SIL 2
EVIDENCE
 Evidence that secondary SW was developed to SIL 2
JUSTIFICATION
 Comparison of the evidence against the requirements for SIL 4
END ARGUMENT
```

As with the software assurance case of the previous section, it is appropriate, now that we have presented this example of a system assurance case, to ask whether it is a sound and persuasive case.

To my mind, it is not. The problem is that the software aspects of the case are not integrated into the main argument. We see this first in [17] where the two premises make different *kinds* of claim: the first one makes a claim about hazards, while the second makes a claim about a method. There may certainly be circumstances where the premises to an argument step rightly address different issues and different kinds of issues (an example is [18], where the assumption premises are about hazards and their severities, while the others are about probabilities), but that does not seem to be so here. Rather, there seems to be a gap in the reasoning: what is intended seems to be a claim that hazards to safe operation have been eliminated or mitigated by methods to be justified further down in the argument. One possibility is that some of those hazards concern software and the means of mitigation is development to specific SILs. Another possibility is that rather than being a source of hazards, software is the means of mitigation for some system hazards, and development to specific SILs provides confidence that the software will perform as required. In either case, the issue is that software hazards need to be related to system hazards and analyzed in that context, not elevated to separate and unmotivated treatment at the top level of the case.

This issue arises again in [19] where the arguments for mitigation of the system hazards make no mention of software. As we do not have a description of the system concerned, we can only guess at its structure, and my guess is that the hazards H2 and H3 relate to subsystems that include software and that the failure rates established by Fault Tree Analysis (FTA) include assumptions about rates of software failure, and these rates are assured by development to specific SILs.

It could be that these details are spelled out in fully developed versions of the JUSTIFICATIONs, for which we have provided only terse placeholders. But that would repudiate the idea that arguments are structured around explicit claims or goals: instead, the claims would only be fully interpreted in the justifications and the structure of claims and subclaims would not provide an accurate overview of the argument.

A revised assurance case in which the system and software aspects are more closely integrated can be represented as follows. First, we combine [17] and [18] and eliminate reference to software hazards at the top level.

```
CONTEXT control-system-example
USES
 control-sys-def-docs, operating-context-docs

DEDUCTIVE ARGUMENT system-safety-revised
CLAIM
 Control system is acceptably safe to operate
ASSUMPTION
 The only system hazards are H1, H2, and H3,
 The worst case severity of H2 is XX,
 The worst case severity of H3 is YY
PREMISES
 Hazard H1 has been eliminated,
 Probability of H2 is below 10E-6 per year,
 Probability of H3 is below 10E-3 per year
JUSTIFICATION
 The system is considered safe because all hazards to its safe
 operation have been eliminated or adequately mitigated
END ARGUMENT
```

Then we revise the final two evidential steps of [19] so that subclaims about software development are used in the fault-tree analysis.

```
EVIDENTIAL ARGUMENT H2-mitigated-revised
CLAIM
 Probability of H2 is below 10E-6 per year
ASSUMPTION
 Primary system SW developed to SIL 4
EVIDENCE
 Fault-tree analysis
JUSTIFICATION
 Explanation why the fault-tree analysis is credible and
 reference to documents that specify allocation of SILs to failure
 rates used in FTA
END ARGUMENT

EVIDENTIAL ARGUMENT H3-mitigated
CLAIM
 Probability of H3 is below 10E-3 per year
ASSUMPTION
 Secondary system SW developed to SIL 2
EVIDENCE
 Fault-tree analysis
JUSTIFICATION
 Explanation why the fault-tree analysis is credible and
 reference to documents that specify allocation of SILs to failure
 rates used in FTA
END ARGUMENT
```

In the section of text associated with Figure 3.2, I advocated that arguments should be presented in "simple form" where the premises to evidential argument steps cite only evidence and interior or reasoning steps cite only subclaims, my reason being that evidential steps are interpreted epistemically, while reasoning steps are interpreted logically. Yet, in the argument steps above we have evidential steps that cite subclaims (about SILs) in combination with evidence (about FTA). A rationale for this is that the subclaims in these evidential steps are used as assumptions: they establish a context or "frame" in which the evidence is interpreted. We still interpret the evidence epistemically, but do so under the stated assumptions, which seems an acceptable and useful extension to "simple form" arguments.

We complete the revised case by eliminating 20 and the first step of 21, while retaining its other two steps. I believe this revision presents the case more clearly than the original. The revised case interprets the software as playing a rôle in the mitigation of system hazards, so that reliability of the software is our main concern. As noted earlier, another interpretation could be that the software is an active source of hazards, so our concern then has to be that its development controls these, and that would require a somewhat different assurance case and further revisions to the argument. We cannot know which interpretation is correct because the example does not supply enough information, but I hope this discussion and the way in which issues were brought to the surface has illustrated some potential benefits in “making the case” for safety through an explicit argument.

Chapter 4

Assurance Case Notations and Tools

In this chapter, we briefly describe the concepts, notations, and tools of two popular methods for organizing and presenting assurance cases (CAE and GSN), and then outline ideas and issues concerning other approaches, notations, and tools.

4.1 CAE: Claims-Argument-Evidence

Much of the early work on structured assurance cases described in Chapter 2 was performed by Adelard LLP [9], which is associated with City University in London, UK. In particular, the formulation of assurance cases in terms of claims, arguments, and evidence was developed there and, over the years, this has been systematized and formalized as a collection of concepts and notations referred to here generically as CAE (the proper name for Adelard's specific methodology and notation is Adelard Safety Case Development, or ASCAD [2]).

The general form of a CAE argument step or "block" is shown in Figure 4.1; this is a graphical representation but its interpretation is very similar to the textual form of argument introduced in the previous chapter. Certainly, the *claim*, *subclaims*, and *justification* are the same in both cases, but the CAE block also includes some elements we have not seen before: namely, the *side-warrant* and the (optional) *backing* and *system info*rmation. These novel elements are derived from a style of argument introduced by Stephen Toulmin [122] that we will examine in more detail in Section 5.1.3. Rather than detour into these topics now, it seems best to proceed using the interpretation that a *side-warrant* is essentially an *assumption*

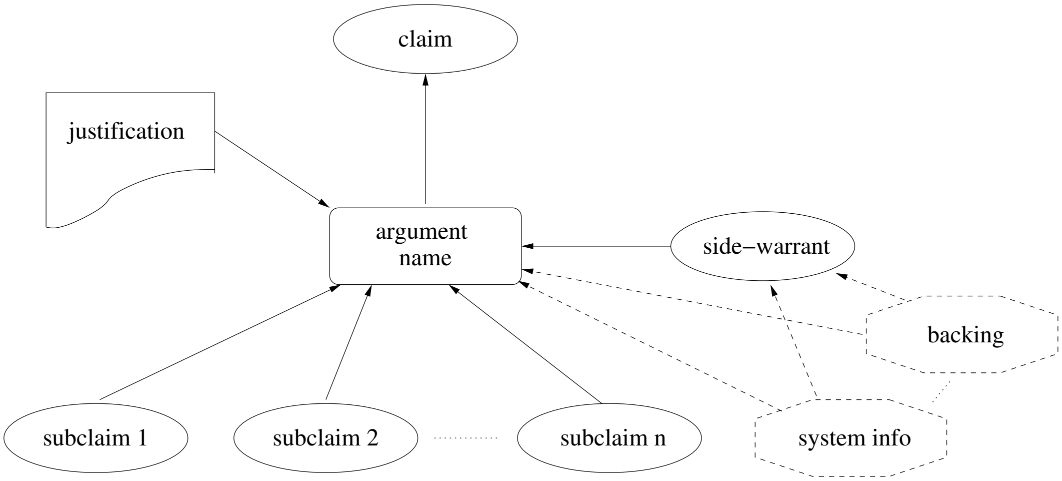


Figure 4.1: Generic CAE Block

(that establishes sufficient conditions for the justification to be sound), and that the *backing* and *system info* are varieties of *confidence* items.[1]

With these interpretations, Figure 4.1 is equivalent to the following textual representation.

```
ARGUMENT argument-name                                                    22
CLAIM
 claim
ASSUMPTIONS
 side-warrant
PREMISES
 subclaim-1,
 subclaim-2,
 ...
 subclaim-n
JUSTIFICATION
 justification
CONFIDENCE
 backing,
 system-info
END ARGUMENT
```

[1] Whereas Figure 4.1 portrays the central node as providing merely the "argument name" with the *justification* as a separate element, CAE usually moves the justification into the central node, which is then called the *argument* with the embedded justification referred to (using Toulmin's terminology) as the *warrant* [12]. These are simply different representations for the same concepts, and further variants are possible; we prefer our form here simply for consistency of presentation.

One of the advantages claimed for a graphical representation such as CAE is that it can represent larger arguments in a form that facilitates comprehension. This is accomplished by chaining argument blocks together, so that the claim of one block becomes a subclaim of another, creating a structure such as that shown in Figure 4.2. Of course, one might wonder how well this scales and whether visual representations do assist the evaluation of large assurance cases. We will examine these topics in Chapter 5. Notice that Figure 4.2 shows only an interior fragment of a larger argument; a full argument would have *evidence* at the bottom, represented by rectangular nodes. Notice also that the argument structure need not be a tree because one subclaim or item of evidence may support more than one argument block.

Most CAE cases are far larger than that shown in Figure 4.2 and it is easy to see that some form of modularity is required. There are two issues here. In reviewing a large assurance case presented as a diagram (or, indeed, in any other form), we clearly need some way to view it at different scales, to collapse subarguments, and to follow references from one node to another; the issue here is one of providing suitable support for rendering and browsing diagrams or other representations of CAE cases. True modularity has more to do with a second issue, which is the ability to develop and review subarguments in relative isolation, while keeping track of what must be revisited when something changes. This issue is mainly concerned with the interpretation and scoping of names and identification of the interfaces that connect different parts of an argument. The same issue is encountered, and adequately addressed, in modern programming and formal specification languages, and one would expect notations for assurance cases to adopt similar mechanisms (possibly "under the hood" of the graphical user interface), rather in the way that our notation in Chapter 3 employed `CONTEXT`s and a "dot" notation to reference claims from other contexts, and `USES` clauses to identify system documents, as illustrated in [10].

Descriptions of CAE do not address these issues; it seems their treatment is delegated to the tools that support CAE. This is unfortunate, as different tools may adopt different solutions and thereby inhibit exchange and interoperation.

4.1.1 CAE Building Blocks

CAE has been in use for many years, so there is much experience in its application and some of this experience has been codified in the form of "building blocks," "templates," and "exemplars" [12]. Building blocks (sometimes called "justification blocks" [120]) refine the generic block of Figure 4.1 into several more specialized instances that represent types of argument commonly used as individual steps within an argument, while templates and exemplars provide guidance on composition of these blocks into larger and complete arguments. Most importantly, the CAE building blocks have been subject to careful analysis (including formal modeling in some

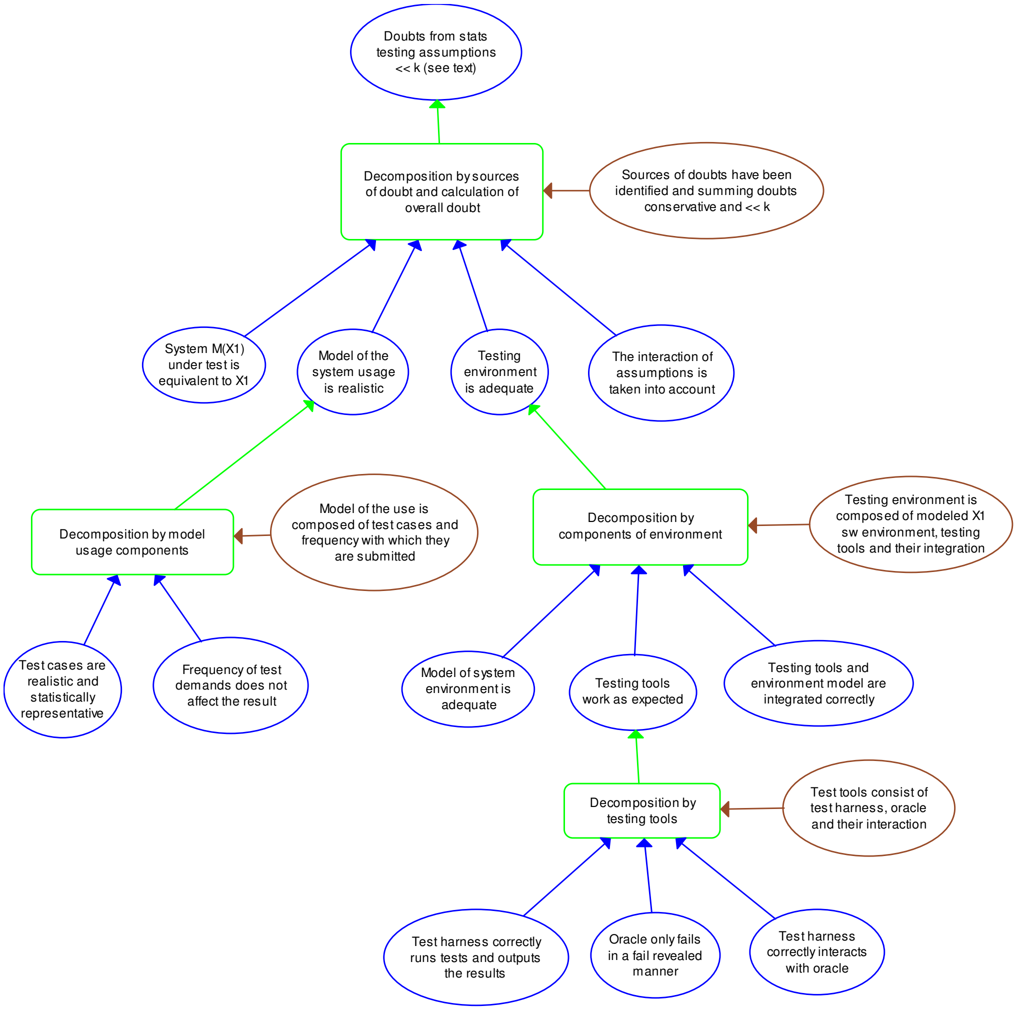

Doubts from stats testing assumptions << k (see text)
Decomposition by sources of doubt and calculation of overall doubt
Sources of doubts have been identified and summing doubts conservative and << k
System M(X1) under test is equivalent to X1
Model of the system usage is realistic
Testing environment is adequate
The interaction of assumptions is taken into account
Decomposition by model usage components
Model of the use is composed of test cases and frequency with which they are submitted
Decomposition by components of environment
Testing environment is composed of modeled X1 sw environment, testing tools and their integration
Test cases are realistic and statistically representative
Frequency of test demands does not affect the result
Model of system environment is adequate
Testing tools work as expected
Testing tools and environment model are integrated correctly
Decomposition by testing tools
Test tools consist of test harness, oracle and their interaction
Test harness correctly runs tests and outputs the results
Oracle only fails in a fail revealed manner
Test harness correctly interacts with oracle

Figure 4.2: CAE Example

cases) and the assumptions required for their use to be sound are recorded in their side-warrants.

For example, we may have some system `X` that is composed of subsystems `X1`, `X2`, ..., `Xn` and we argue that `X` satisfies claim `C`, which we denote `C(X)`, by showing that each of its subsystems also satisfies `C`, that is, we use subclaims `C(X1)`, `C(X2)`, ..., `C(Xn)`. We might use this argument step to claim that a software system will generate no runtime exceptions by showing it to be true for each of its software components. However, this type of argument is not always sound—for example, we cannot argue that an airplane is safe by arguing that its wheels are safe, its rudder is safe, ...and its wings are safe. Soundness is contingent on the property C, the nature of the system X, and the way in which the subsystems X1, X2, ..., Xn are composed to form X. Furthermore, claim C(X) may not follow simply from the same claim applied to the subsystems, but from different subclaims applied to each: C1(X1), C2(X2), ..., Cn(Xn). For example, a system may satisfy a timing constraint of 10ms. if its first subsystem satisfies a constraint of 3ms., its second satisfies 4ms. and its third and last satisfies 2ms. (together with some assumptions about the timing properties of the mechanism that binds these subsystems together).

Thus, we can see that unrestricted "composition" (or "decomposition," depending which way you look at it—CAE uses the latter term), is too general to serve as a building block: it needs to be refined into more specialized cases. Adelard provide several examples of such refined "decomposition blocks" [10], which are paraphrased below. The side warrant to each of these blocks must ensure that the conjunction of subclaims implies the claim; in cases where the subclaims and claim concern the same property *P*, this generally follows if *P* distributes over the components and the mechanism of decomposition.

Component: This is the simplest case considered above: a claim about a system is established by demonstrating the same claim about its components.

Architecture: This is a more sophisticated form of component decomposition: the system claim is established by showing that the architecture establishes certain subclaims and the components establish others. For example, a partitioning architecture may establish subclaims about fault propagation across components, thereby allowing subclaims about components to assume a relatively benign environment.

Property: Here, it is not the system but the claim that is decomposed: a claim C is refined as a conjunction of subclaims C1 AND C2 AND ... AND Cn and the claim is established by showing that each subclaim holds. For example, the property of security may be decomposed into confidentiality, integrity, and availability.

Environment: Here it is the system environment that is decomposed and the system claim is established by showing that it holds in each environment. For example, claims about an aircraft system might be shown to hold when the aircraft is on the ground, in the air, and in transition between these.

Function: The function of the system (i.e., what it does) is decomposed into subfunctions and suitable subclaims established for each.

Mode: The behavior of the system is decomposed into separate modes (e.g., takeoff, climb, cruise, etc.) and suitable subclaims established for each.

Configuration: The system exists in different configurations and the claim is established for each one.

Hazard: This is a surprising absence from Adelard's list. One of the most common ways to structure an assurance argument is to identify all hazards to the safe operation of the system and then decompose the argument into subclaims that the system is safe for (or adequately mitigates) each hazard, and their combinations.

It is possible that Adelard view this as an instance of environment decomposition, because they mention that an argument could be decomposed into the different kinds of threat posed by the environment.

Phases: Here, the decomposition is on the evolution of the system over time. For example, we could decompose a claim about a property of the system into subclaims that the property is true of the system when it initializes, and remains true thereafter. For assurance cases that consider the risks posed over the total lifetime of a system (e.g., a nuclear reactor), we might decompose the claim into a subclaim about the safety of the system in operation, and another subclaim about the safety of its disposal plan.

In addition to those based on various kinds of *decomposition*, Adelard identify four additional types of building block, which we sketch below.

Substitution: this is used to argue that a claim for one system follows because an "equivalent" claim has been established for a different but "substantially equivalent" system. This style of argument is used for medical devices under the "510(k) Premarket Notification" procedure of the FDA [73], but it might also be used to argue that a claim remains true following some change or upgrade to the system or its method of manufacture. A different application of this building block is to argue that a property derived by analysis of a model also holds for the real system, because the model and the system are adequately "equivalent" for this purpose.

Concretion: this is used to give more precise interpretation to a claim. It is particularly relevant to regulatory regimes that employ an ALARP ("As Low As Reasonably Practicable") principle [95]. For example, the unqualified claim that a system is "safe" may be replaced by a more quantitative claim about the probability of various kinds of loss events or failure conditions. Alternatively, the claim that a system is secure might be replaced by the claim that it is secure against specific threats.

Evidence Incorporation: this building block corresponds to what we call an evidential step; it is used "to demonstrate that a subclaim is directly satisfied by its supporting evidence" [12].

Calculation: this is used to argue that a claim follows by calculation from other claims. The notion of "calculation" might be a conventional numerical one (e.g., average time of data retrieval is computed from its time of retrieval from the cache vs. from memory, and its probability of being in the cache), but it could also be interpreted as a mechanized (or mechanically checked) formal verification of some property of a model or design.

CAE building blocks provide significant guidance in the construction of assurance case arguments; at each step, consideration of the repertoire of standard building blocks is likely to suggest a suitable move. Although few details are given, recent CAE documents suggest how building blocks may be combined to yield complete arguments [10,120]. For example, the argument of Figure 4.2 (taken from [120, page 41]) is built from four different decomposition blocks.

In contrast to the incremental, one step at a time, approach to the construction of arguments encouraged by CAE building blocks, CAE "templates" and GSN "patterns" (discussed in Section 4.2.1), focus directly on complete arguments. Substantial examples are available from Adelard at `http://www.adelard.com/services/WOMESafetyEnvCaseTmplt/index.html`, where they provide templates for "weapons, ordnance, munitions and explosives" procured and maintained by the UK Weapons Operating Centre, together with an approved "code of practice" for their use [134]. The templates provide guidance not only for the construction of assurance cases for safety and environmental safety of these kinds of systems, but also their review.

Observe that whereas building blocks each concern a single argument step and their side-warrants can accurately document the conditions for applying them soundly, it is much harder to precisely characterize the assumed context and intended application for larger units of argument, such as templates. However, it is plausible that software assurance cases (where the top claim is correctness, not safety) exemplify a sufficiently narrow (i.e., well characterized) application that sound templates can be provided for complete arguments. Sound use of templates in more general applications seems dependent on human judgement, and suitable guidance would be welcome. Work on "argument schemes" by Yuan and Kelly [135] proposes guidance in the form of "critical questions" to be considered when instantiating an argument scheme.

In my opinion, all these generic "outlines" for partial or complete arguments have the utmost value. It is not easy to develop or to review the argument of an assurance case, and use of community-validated outlines complete with side warrants, well-characterized scope, and critical questions is to be encouraged. We return to this topic in the Conclusion, Chapter 6.

4.1.2 CAE Tools

CAE is supported by Adelard's commercial *Assurance and Safety Case Environment* (ASCE, pronounced "ace") [37].[2] This can render arguments in both CAE and GSN notations and can export documents in Word, html, and pdf formats. A free (but Windows-only) *ASCE-Browser* provides capabilities for viewing "HTML exported ASCE networks" beyond those available in a standard web browser. ASCE is a comprehensive tool that is widely used in industry and, despite its origins in CAE, it is said to be the most widely used tool for GSN. ASCE supports modular assurance cases in both CAE and GSN (apparently based on the GSN extensions described in Section 4.2.1) and can access external data sources such as system documentation, hazard logs, and other ASCE networks, and it provides a form of version control over these. An entire assurance case represented in ASCE can be exported as an XML document and imported elsewhere. Regular "ASCE User Group" meetings provide a forum for information exchange and presentations by both developers and users.

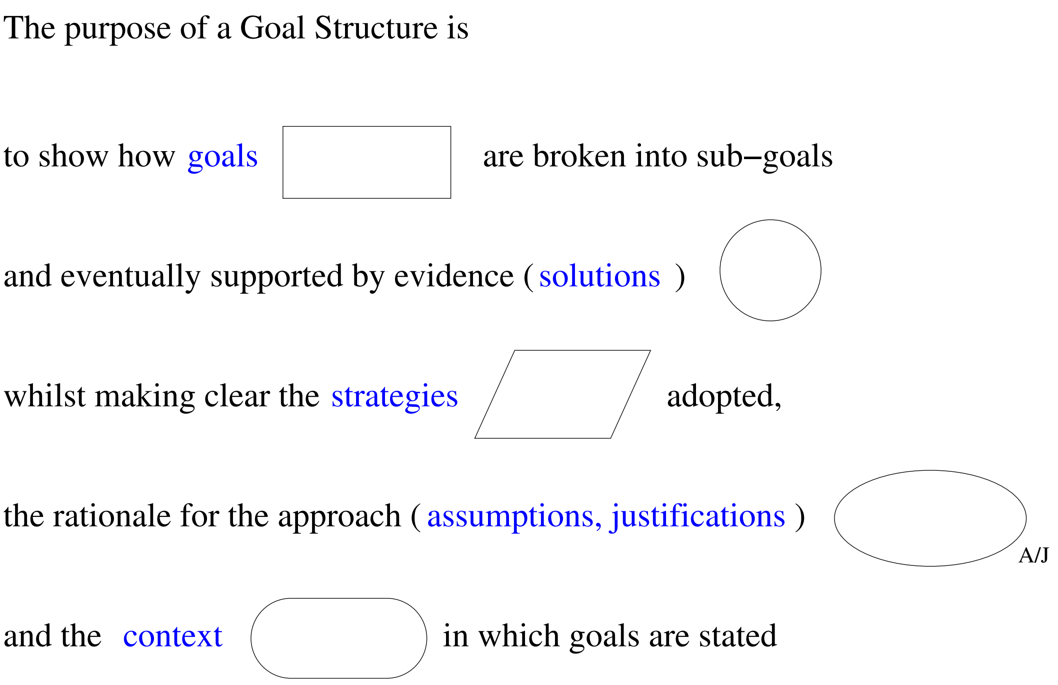
The notations supported by ASCE are provided by means of "schemas" that can be extended or developed by users; schemas determine both the internal structure (represented in XML) and the way it is rendered. The slides for one presentation contain an intriguing reference to "challenge schema" [36] but no details are given. The analysis and reporting capabilities of ASCE can likewise be extended with "plugins." Thus, a schema can (indeed, has been) written to represent fault trees in ASCE and a plugin can then perform an analysis based on semantics it ascribes to that representation. ASCE itself seems not to apply any interpretation to its representation of arguments. Some experimental plugins do perform "quantitative confidence propagation" [36] but this is based on simple propagation of structural information (namely, the extent to which the evidential leaves of each subargument are populated).

4.2 GSN: Goal Structuring Notation

The University of York was an important contributor to the development of structured assurance cases [130]. The Goal Structuring Notation (GSN) was developed there in the 1990s [131], most particularly in the doctoral thesis of Tim Kelly [76]. Like CAE, it has evolved over time, mostly through simplification. It is a graphical notation built on five basic node types as described in Figure 4.3, which is taken from [32]. Observe that the shape of each node indicates its type.

With respect to the terminology introduced in Chapter 3, GSN *goals* correspond to claims, *solutions* to evidence, and *strategies* to arguments. The kind of argument

[2] There seems to be no readily accessible paper that describes modern versions of ASCE; however, some documentation is available at the web site http://www.adelard.com/asce/.



The purpose of a Goal Structure is

to show how goals are broken into sub–goals

and eventually supported by evidence (solutions)

whilst making clear the strategies adopted,

the rationale for the approach (assumptions, justifications)

and the context in which goals are stated

Figure 4.3: GSN Elements

employed (e.g., an argument over hazards, or over components) is written inside the strategy box, so that GSN strategies typically behave like CAE decomposition blocks (these were described in Section 4.1.1). The interpretations of GSN *assumptions* and *justifications* correspond to the same terms in Chapter 3. Note that *assumptions* and *justifications* in GSN are both represented by elliptically shaped nodes, so a subscripted A or J is used to distinguish them. The GSN notion of *context* is currently subject to debate and is described later.

A GSN example is displayed in Figure 4.4; the textual assurance case presented in [17] and subsequent boxes in Section 3.2 was based on this example, which is from the GSN Community Standard [48]. A diagram such as Figure 4.4 is referred to as a "Goal Structure."

There are obvious similarities between GSN and CAE, but also some differences. For example, GSN often connects goals to subgoals without an intervening strategy (i.e., argument step) to justify this: for example, the top goal G1 in Figure 4.4 is directly connected to the two subgoals G2 and G3. The explanation seems to be that GSN generally assumes that goals are related to subgoals by some kind of

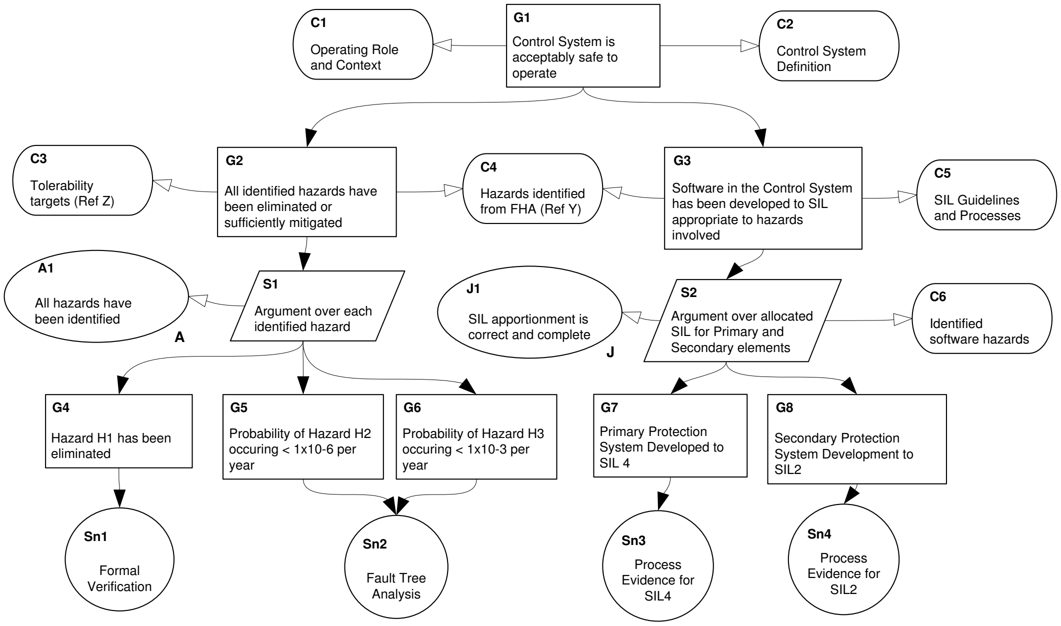

G1
Control System is acceptably safe to operate
C1
Operating Role and Context
C2
Control System Definition
C3
Tolerability targets (Ref Z)
G2
All identified hazards have been eliminated or sufficiently mitigated
C4
Hazards identified from FHA (Ref Y)
G3
Software in the Control System has been developed to SIL appropriate to hazards involved
C5
SIL Guidelines and Processes
A1
All hazards have been identified
A
S1
Argument over each identified hazard
J1
SIL apportionment is correct and complete
J
S2
Argument over allocated SIL for Primary and Secondary elements
C6
Identified software hazards
G4
Hazard H1 has been eliminated
G5
Probability of Hazard H2 occuring < 1x10-6 per year
G6
Probability of Hazard H3 occuring < 1x10-3 per year
G7
Primary Protection System Developed to SIL 4
G8
Secondary Protection System Development to SIL2
Sn1
Formal Verification
Sn2
Fault Tree Analysis
Sn3
Process Evidence for SIL4
Sn4
Process Evidence for SIL2

Figure 4.4: GSN Example

decomposition, and it is unnecessary to provide a strategy to describe this when it is suitably obvious. The language of the GSN Community Standard seems to suggest that providing a strategy to explain the decomposition of goals into subgoals is the exception rather than the rule.

> "When documenting how claims are said to be supported by sub-claims, it can be useful to document the reasoning step—i.e., the nature of the argument that connects the claim to its sub-claims. This is done in GSN by documenting the strategy of the argument which links goals" [48, Section 0.4.4].

Similarly, all the solutions at the bottom of the figure are directly connected to their parent subgoals without intervening strategies.

Furthermore, it seems that providing a justification for the application of a strategy is optional.

> "Argument authors may feel the need to justify a particular claim or argument strategy, to provide some explanation as to why they consider it acceptable. This is achieved in GSN by the use of the justification element" [48, Section 0.4.7].

Neither the principles described in Chapter 3 nor CAE countenance omission of strategies (cf. arguments) or justifications (CAE does allow evidence to be directly connected to subclaims without intervening arguments, although this is deprecated). However, these omissions in GSN can be seen as simple abbreviations: some standard "default" strategy and justification is implicitly inserted whenever these nodes are missing. The danger, of course, is that the default strategy and justification are not documented, so different stakeholders may mentally supply different interpretations. And, as we saw in the discussion of CAE decomposition blocks in the previous section, some decompositions require fairly complex assumptions.

In addition to permitting *no* strategy between a goal and its subgoals, GSN also allows *multiple* strategies. The idea is that "both lines of argument are required to support the goal" [48, Section 1.3.3]. It seems that this also can be interpreted as an abbreviation: one that elides the decomposition (and corresponding strategy and justification) of the original top goal into two intermediate subgoals. In my opinion, it would be better to avoid all these elisions (of strategies, justifications, and subgoals), or to at least specify precisely how they are to be interpreted.

The example in Figure 4.4 uses two different kinds of arrowheads: solid and open. The former indicate "*SupportedBy*, that is, inferential or evidential, relationships between elements," while the latter indicate "*InContextOf*, that is contextual relationships" [48]. These distinctions seem redundant: solid arrowheads always point to goal, solution, or strategy nodes, while open ones always point to assumptions, justifications, or context nodes; hence, no additional information is conveyed by the different arrowheads.

We now return to the interpretation of *context* nodes in GSN. Graydon [46] provides a very careful examination of three different interpretations for this element and observes that a mismatch between the interpretations applied by different stakeholders can have serious consequences. Graydon specifically considers the node `C2` "`control system definition`" that provides context to the goal `G1` "`control system is acceptably safe to operate`" in Figure 4.4. One interpretation is that the context is asserting a claim that the system as operated matches the definition stated in the context node. I think this interpretation is better represented by an assumption. A second interpretation is that the context is providing additional information; it seems to me that this also is better represented as an assumption or additional premise. The third interpretation, and the one advocated by Graydon, is that contexts provide "explications": that is "the text in a context element must be of the form X: Y where X is a phrase and Y is its explication. . . Y should identify relevant documentation where appropriate. . . Arguments should be understood as if explicated terms were replaced by their explications." This interpretation of GSN context nodes coincides with that of the `MEANS` clause of the textual language used in the previous section.

GSN diagrams often contain a large number of context nodes that can obscure the structure of the argument (see Figure 4.4, for example, where context nodes comprise more than a third of the total). If Graydon's recommended interpretation is adopted, then it seems that context nodes can simply be eliminated from the diagrams. Text in all GSN nodes is terse and surely requires amplification or "explication" elsewhere. In a graphical environment, we might expect that hypertext links would lead to this explication and there is no need for a special node in the diagram.

4.2.1 Modularity and Patterns in GSN

As discussed for CAE, large assurance cases demand some form of modularity. GSN provides this by allowing nodes to be decorated with symbols (typically a "file folder" icon) that indicate an "away" node: decoration at the top means this node is being defined in the local document and can be referenced from other documents, while decoration at the bottom indicates this is a reference to a node defined elsewhere. Finally, a small diamond can be used to represent part of a goal structure that is yet to be developed. Other than away nodes, names are presumably local to the document in which they appear, but these and related language issues (recall, for example, the discussion on page 38 of the logical context in which claims or goals are interpreted) are not described in GSN reference documents.

There has been much work on "patterns" for common styles of argument in GSN, rather like blocks and templates for CAE. Kelly's doctoral thesis [76] includes a collection of patterns derived from examples; Weaver's Thesis [128] develops patterns for software assurance that are designed to be connected together to compose a larger argument from the individual patterns; and Ye's thesis [133] considers systems with COTS components and provides patterns for arguing that evidence is adequate to support a claim at a given assurance level.

More recent work describes patterns for software safety arguments [54] and their evaluation on two case studies [53]. Whereas the CAE building blocks outlined in Section Section 4.1.1 focus on individual argument steps (*strategies* in GSN terms), the GSN software safety argument patterns address complete safety arguments (for aircraft software this would include topics covered by ARP 4761 and ARP 4754A in addition to those of DO-178C). There are five patterns, which are sketched in [54] and presented in detail in [83, pp. 150–178]. Paraphrasing [54], the patterns are as follows.

High-Level Software Safety Argument Pattern: This integrates the patterns below.

Software Safety Contribution Pattern: This pattern assumes a layered software development process and focuses on management of system hazards introduced at each layer or "tier."

Derived Requirements Pattern: This pattern focuses on assurance that "derived requirements" (recall Section 3.1.1) are adequately captured at each tier.

Hazardous Contribution Pattern: This focuses on the argument that all potentially hazardous failures are captured by "derived requirements." (The relationship between this pattern and the one above is unclear to me.)

Strategy Justification Pattern: This concerns the argument that the strategy adopted in a software safety argument provides adequate confidence in the claim.

The evaluation study [53] proposed four "assurance considerations" for each "tier" of a layered software development process. Paraphrased and reordered these are as follows.

1. Safety requirements at each tier are appropriate for the design at that tier and traceable to higher tiers.
2. Each tier satisfies its safety requirements (i.e., it is correct with respect to these requirements).
3. Each tier introduces no hazardous flaws.
4. Hazardous failure behavior at each tier is identified and mitigated.

Although I have reservations about these particular patterns and the evaluation considerations (e.g., why are these not goals?), the general direction seems a good one: it attempts to link sound practices for developing safe systems with arguments for their assurance. The rationale for these particular patterns is derived from the "six step method" for constructing arguments described in [76]. A somewhat similar idea of "assurance-based development" is proposed in [45] and it would be valuable to develop a rigorous link between these process-oriented development methods and the static description provided by an assurance case template.

The DEOS project, which is outlined in Section 4.3, has developed and implemented a language that can describe GSN patterns and instantiate them appropriately, and has applied it to many of the patterns found in the literature [82].

4.2.2 Assured Safety Arguments

As we saw in Chapter 3, some of the subclaims in an argument may directly support the parent claim, while others provide "confidence" in some aspect of the argument. And, as we also saw (and will examine again in the next chapter), the appropriate use of confidence claims, and their logical interpretation, are difficult topics. A paper by Hawkins *et al* [55] makes a significant proposal on the appropriate use of confidence claims and suggests suitable modifications to GSN that seem to be gaining acceptance. Hawkins *et al* observe that a typical goal structure mixes two kinds of arguments and claims: those *about safety* and those about *confidence in the argument for safety*. Almost any argument step or item of evidence may be deemed relevant under one or other of these headings, leading to a tendency to include them all "just in case," resulting in "voluminous, rambling, ad infinitum arguments" [55] that fail to fulfill the primary purpose of an assurance case, which is to communicate a clear argument.

Hawkins *et al* [55] propose that a safety case should provide an "assured safety argument" in which the primary argument is solely concerned with safety (or the topic of the top-level claim, in the case of an assurance case about something other than safety). They suggest that this argument should be focused on the identification and mitigation of hazards associated with the system. I consider this recommendation rather too specific as there are other ways to argue safety than by decomposition over hazards, but let us accept it for the time being. The safety argument is composed of various types of node such as contexts, assumptions, subgoals, solutions, and strategies, and the essence of Hawkins *et al*'s proposal is that confidence claims should be attached to some of these rather than included in the general goal structure. Specifically, they propose that each (group of) confidence claims should be associated with some particular link in the goal structure of the safety argument, referred to as its *Assurance Claim Point* (ACP), and indicated by a black square on the corresponding link as illustrated in Figure 4.5.

Hawkins *et al* limit ACPs to links incoming to (i.e., referring to) strategies, contexts, and solutions (cf. ACP1, ACP2, and ACP3, respectively, in Figure 4.5).

Confidence claims attached to a link incoming to a strategy yield an *asserted inference* and the purpose of such claims is to justify the inference associated with the strategy (i.e., to make the case that the goal follows from the subgoals). In my opinion, this is the rôle of the *justification* that is (optionally) attached to each strategy, and it is a mistake to conflate this with a confidence claim. In particular, they are different kinds of objects: a justification provides a warrant, whereas a claim is a predicate.

Confidence claims attached to a link incoming to a context yield an *asserted context* but, in light of Graydon's proposal that GSN contexts should be interpreted as explications (recall the discussion on page 65), I see no purpose in such ACPs: an explication simply defines what a term means—hence, it cannot be right or wrong and is not subject to confidence.

Confidence claims attached to a link incoming to a solution yield an *asserted solution* and serve to support the claim that the cited evidence establishes the associated goal. As explained in the text associated with Figure 3.2 on Page 41, multiple items of evidence are interpreted "cumulatively," and I agree that confidence claims

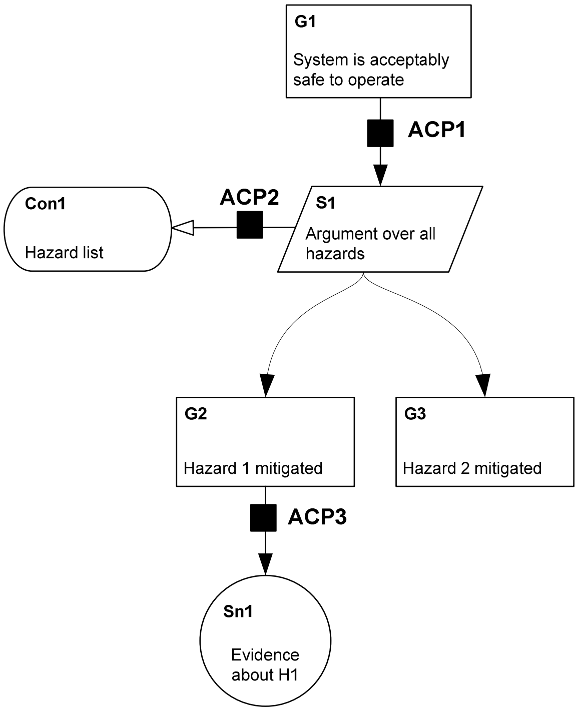


Figure 4.5: Assurance Claim Points in GSN

can be added to this accumulation (how this may be done is explained in Section 5.3.1).

Thus, I agree with the recommendation of [55] that confidence claims should not be part of the general claim or goal structure, but disagree with their proposal for “assured safety arguments” and reject two of their three specific kinds of “claim points”; in my opinion, confidence claims are useful (and can be given a semantics) only in support of evidential steps (i.e., asserted solutions).

4.2.3 GSN Tools

The Website that hosts the GSN Standard [48] lists several tools implementing GSN,[3] but most of these no longer seem to be available. Here, we briefly outline five tools that are currently available.

Although initially developed for CAE, Adelard’s ASCE is said to be the most widely used GSN tool. As we described in Section 4.1.2, ASCE can be customized by means of schemas and plugins, and a schema that supports GSN is a standard

[3] See http://www.goalstructuringnotation.info/archives/41.

part of the system. All the capabilities of ASCE apply to GSN in the same way as to CAE.

Another tool from the UK is ISCaDE (Integrated Safety Case Development Environment), which is a product of a company called RCM2. The tool and its documentation are available at http://www.iscade.co.uk/, together with its specification [94] and an example of its use for a railway application [40]. A unique attribute of ISCaDE is said to be its integration with DOORS, which is a widely used tool for the management of requirements.

Astah is a Japanese company whose tools for UML and SysML are widely used, particularly in Japan where they are said to be the market leaders. They have recently released *Astah GSN*, which is available at http://astah.net/editions/gsn. One novelty of this system is that it supports dialectical examination of an argument with *pro* and *con* subarguments using the approach of Takai and Kido [119] that is outlined in the next section.

The *Safety Case Toolkit* (SCT) from Dependable Computing is a comprehensive system for development, management, and presentation of assurance cases [4]. Information about SCT, and also about the less comprehensive *Argument Editor* and free *GSN Editor* are available at http://www.dependablecomputing.com/tools/index.html. SCT is intended to support large assurance cases and builds on standard tools for version control and workflow management. Most GSN tools provide some means to render a Goal Structure as text or html for external review or as documentation but SCT seems unique in that it can render a complete assurance case as a standalone web site with extensive hyperlinking between the Goal Structure and its supporting assets (e.g., items of evidence and documentation) and sophisticated functions for browsing in different levels of detail. Whereas other tools likely require reviewers to have access to the tool itself, the website generated by SCT may provide adequate functionality for this purpose.

The previous tools are commercial; *AdvoCATE* (the Assurance Case Automation ToolsEt) is a research tool developed at NASA Ames Research Center [30]. Although it provides an Eclipse-based environment for constructing and reviewing GSN diagrams "by hand," its primary purpose is to explore various kinds of automated support. These include the derivation of assurance cases from automated proofs [27], and methods to assist comprehension of large assurance cases by extracting views based on queries [26], and hierarchy [31]. Another NASA-sponsored tool is *CertWare* from Kestrel Technology, which is described as a "workbench to develop, maintain, and analyze safety cases." There is little documentation at present but the system is available at http://nasa.github.io/CertWare/.

AutoFocus is a model-based development platform that originally came from the Technical University of Munich; the latest version, known as AF3, is developed by its spinoff institute Fortiss. AF3 provides substantial support for integrating model-based development with construction of safety cases, which it can render in GSN [125]. AF3 is available under the Eclipse Public License.

4.3 Other Approaches, Notations, and Tools

Here we describe approaches, notations, and tools that are related to assurance cases but do not fall under either of the two traditions considered so far.

4.3.1 DEOS and Related Work in Japan

Much work on assurance cases has been accomplished in Japan as part of the DEOS project (Dependable Operating Systems for Embedded Systems Aiming at Practical Applications), which ran from 2008 to 2013. The overall ambition of DEOS is large: it is to ensure the dependability of open systems subject to change [121]. In DEOS, a system assurance case is maintained as part of the system and is used to guide its adaptation to failure and to changed requirements. For local adaptation (e.g., responding to failure), an online representation of the assurance case (called a D-Case) is used to guide adjustments to the system, potentially automatically, under the guidance of a scripting language call D-Script and associated tools [78].

D-Case assurance cases are developed and maintained with the D-Case editor [25] and its associated tools, which include an assurance case language [81] and a library of patterns [82]. The language resembles a functional programming language, but is based on GSN and assurance cases can be created and rendered in GSN using the D-Case editor (which is an Eclipse plugin). An interesting feature of the language is its support for patterns, which can be defined generically and instantiated with appropriate replication (e.g., a pattern for argument over hazards, called Hazard Avoidance Pattern, can be instantiated with as many hazards as necessary). Support for the language includes an extensive pattern library, whose members are based on examples described in the GSN literature. There is also an experimental connection to the Agda theorem prover, but apparently no other automated or interactive support for challenging or evaluating an assurance case.

Many papers from DEOS and its related and descendent projects are only now reaching publication. One such proposes a GSN extension to support dialectical examination of assurance cases [119]. The idea is that additional subarguments can be attached to the nodes of an existing argument and labeled either *pro* or *con*: the former supports the node concerned while the latter challenges it. *Con* arguments are related to the idea of *defeaters*, which will be introduced in Section 5.1.2, and *con* arguments may themselves be challenged by *con*s to their own components or *pro*s to the nodes that they attack. Rules are defined for “adding up” *pro* and *con* subarguments and deriving an evaluation for confidence in the overall argument;

again, this is related to ideas in defeasible logic and eliminative induction that will be discussed in Sections 5.1.2 and 5.1.4.

4.3.2 Interchange Formats

The basic elements of assurance cases are few and simple: claims (or goals), evidence (or solutions), and arguments (or strategies). The specific differences between CAE and GSN are more in matters of emphasis and style than fundamentals. Thus, it is not unreasonable to suppose that tools supporting different methods and notations could, nonetheless, share a common underlying representation. Such a representation, known as the Structured Assurance Case Metamodel (SACM) has been developed under the auspices of the Object Management Group (OMG) [112]. SACM comprises two formerly separate parts: an Argumentation Metamodel (ARM) and a Software Assurance Evidence Metamodel (SAEM). The models are defined using the MOF (Meta-Object Facility), XML (Extensible Markup Language), and XMI (XML Metadata Interchange) formats of OMG. Several tools, including ASCE, SCT, Astah GSM, and AdvoCATE, are able to import and export assurance cases represented in SACM.

Although its initial purpose was to support existing notations for assurance cases, SACM goes beyond CAE and GSN in some respects. For example, one or more claims can be declared to challenge another claim, thereby providing a mechanism for exploring dialectical challenge to an argument. Observe that this mechanism is different to that of Takai and Kido described above: in their proposal, a claim is declared to be *pro* or *con* another claim (i.e., the annotation is associated with the first claim), whereas in SACM the annotation is attached to the relation between the two claims, not to the claims themselves.

It should be noted that the academic field of *Argumentation* (mentioned in Section 5.1.4) has an Argument Interchange Format (AIF) whose draft specification [16] has more than 200 citations. A website for argument interchange using AIF is provided at `http://www.argumentinterchange.org` and a list of tools that can use AIF is available at `http://www.argumentinterchange.org/library`. AIF is intended to support models of human arguments and arguments in contested domains, so it can represent defeasible and dialectical reasoning; on the other hand, it has little support for representing the provenance of evidence, which has rich support in SACM.

4.3.3 Relation to Formal Methods

Assurance cases have something in common with formal methods insofar as both concern reasoning, so it is worth examining their relationship in a little more detail. Modern formal methods are essentially synonymous with formal verification, where methods of automated deduction (i.e., automated or interactive theorem proving, model checking, abstract interpretation, and related techniques) are used to prove or verify properties (e.g., invariants) of a formal system description, or a relationship between one formal description and another (e.g., that a program implements a specification). An assurance case argument whose reasoning or interior steps are deductively sound is likewise a proof of its top claim from its evidence. This similarity may prompt some to suggest that the proof extracted from a formal verification can form part of an assurance case, or that the methods of formal verification can be used to evaluate the argument of an assurance case or to automate its construction. In their simple forms, these suggestions are mistaken in my view, but there are ways in which assurance cases and formal methods can be combined productively.

In my view, the automated deduction performed as part of formal verification should be viewed as a calculation, just like the computational fluid dynamics performed as part of the analysis of an airfoil: it is an atomic item of evidence. Proof steps and internal claims used in performing the formal verification are unlikely to be of interest at the level of the assurance case and so it is not useful to generate parts of the assurance case argument by extracting these from the verification. Instead, the assurance case should focus on the credibility of the formal verification: why do we believe that the formally verified property (a string of symbols in some mechanized logic) bears the interpretation (presumably some real-world claim stated in natural language) used in the assurance case; similarly, why do we believe that the verified system model (e.g., a Simulink diagram, or a state machine expressed in the language of a model checker) accurately represents the corresponding real-world artifact (e.g., a C program, or its compiled EOC); and why do we trust the formal verification (i.e., the automated deduction that is employed, and the many translations from one representation to another that are typically performed inside the verification system)?

Dually, I am skeptical of the value in using the techniques of formal verification (or, indeed, any automated method) to generate parts of an assurance case. The essence of an assurance case is that its argument is the focus of human review, and it therefore seems that its construction should likewise be the product of human consideration. There may, however, be value in automatically generating subclaims, and even outline arguments, for standardized activities—but the templates for these (and, indeed, they would be very much like the building blocks, templates, and patterns described in Sections 4.1.1 and 4.2.1) should be products of human construction rather than automation.

4.3.4 Assurance Case Workflows

One area where I believe that integration of assurance cases with formal—or, at least, logic-driven—methods could have great value is in managing and maintaining an assurance case during system development and evolution. The assurance case should not be an afterthought but an integral part of system development—possibly even its driver [45]—and there will be many items of evidence associated with the assurance case that need to be revised as system development proceeds (e.g., hazard logs and analyses, documentation, tests, reviews, formal verifications). Software systems are generally developed in environments with automated version control (e.g., `git`) and build systems (e.g., `make`) and it is feasible for an assurance case tool to use these to manage the regeneration of evidence as necessary—SCT, for example, does this.

Conventional build tools like `make` have software development as their focus and keep track of linear dependencies among components—so that if a component changes, those that use it must be rebuilt, and so on transitively to the target of the build (e.g., "`make main`"). In building or updating an assurance case, however, the target is to (re)establish truth of a claim, and propagation of linear dependencies is not the best way to do this: for example, if the claims associated with a component remain true following changes to its design, there is no need to reconsider the claims higher up the argument. Thus, the build process for an assurance case needs to be based on the logic of its argument. It may be more complicated than this, however: some items of evidence that correspond to atomic claims in the assurance argument may be the products of complex workflows (e.g., a counterexample-guided abstraction refinement loop for formal verification, construction and analysis of fault-trees, or an extensive set of tests). Hence, the build process that supports an assurance case should be driven by a logic-based specification of workflows. Some assurance case tools provide for this: for example, SCT uses BPMN2 (Business Process Modelling Notation 2) [14] and the *Activiti* open source workflow engine.

The *Evidential Tool Bus* (ETB) from SRI [21] takes this further: as its name suggests, the ETB is a framework that enables multiple tools to collaborate in the construction and management of evidence. It uses an extension of Datalog as its workflow language [22]; Datalog is based on Horn clauses and, as we explained earlier (Section 3.1), assurance case case arguments naturally take the form of Horn clauses, so the ETB is able to manage the workflows for construction of complete assurance cases, and is used experimentally by one aerospace company in the management of workflows for DO-178C. Whereas a software build process generally involves at most two platforms (i.e., development and target), an assurance case may involve many (e.g., tests may require a specialized platform with simulated or hardware-in-the-loop sensors and actuators, while verification may require high-performance machines with specialized software), and some of them may require human participation (e.g., a manual review). Thus, the ETB is fully distributed and can invoke and coordinate activity on multiple platforms; an evidence-producing activity is connected to the ETB by means of a "wrapper," and for a human-mediated activity the effect of the wrapper may be to send an email (e.g., "please review the following change to HLR xxx and update database zzz accordingly"). The results of workflow-initiated activities are recorded in a “claims table” from which the current state of the assurance case can be derived.

Chapter 5

Assurance Case Evaluation

We have introduced and illustrated the general principles of assurance cases and outlined the notations commonly used to represent them, and we now turn to the question of how to determine whether an assurance case instills sufficient confidence to justify deployment of the system under consideration.

There are several difficult topics here. One is the relationship between confidence in a case and the decision to deploy the system. In most regulated industries, the decision to deploy is based on some quantified estimate of risk which, for the time being, we can think of as a probability of (safety-critical) failure. Confidence in an assurance case also can be expressed as a probability, but it is different kind of probability than the one that speaks of failure (subjective rather than frequential), and it is about a different thing, indeed a different *kind* of thing (the case rather than the system), so how can these be related? We examine this question in Section 5.4.

Related to this is the question how to quantify confidence in a case. This will involve some combination of logic and probability, which is a notoriously difficult topic (because the two fields have different semantic foundations). We examine this topic in Section 5.3.

The topics above assume that the assurance case is, in some sense, *sound* and we are measuring its quality. But how can we tell if a case is sound or not, and what is the difference between an unsound case and a sound one in which we have low confidence? What is going on when we deliberately weaken a case, as is done for the lower Software Levels in DO-178C? We examine these topics in Section 5.2.

Finally, what exactly is an assurance case? It is described as providing an argument that evidence about the system supports a significant claim about it, but what is an argument, and how does it differ from a proof?

The following sections consider these topics in the reverse order to their introduction above. We begin by discussing argumentation and the nature of assurance cases.

5.1 Assurance Cases and Argumentation

In informal speech, the word “argument” is used in several ways that are also relevant to assurance. One way carries the sense that an argument is a way of getting at the truth (e.g., “That’s a valid argument!”), another invokes the to-and-fro, or dialectical, nature of argument (e.g., “We argued our way to a satisfactory conclusion.”), while a third stresses the rhetorical or persuasive content of argument (e.g., “That’s a very good argument!”).

Since the time of the ancient Greeks, separate subfields of philosophy have developed around each of these different topics. To get at the truth, we need to apply sound reasoning to solid facts; sound reasoning is the subject of *logic*, while solid facts or, more fundamentally, the question of what it is to *know* something, is the subject of *epistemology*. The dialectical element has traditionally been subsumed under study of the methods of philosophy (e.g., “the Socratic Method” or “Scientific Method”), but lately has become (a large part of) a distinct field known as *argumentation*. Persuasion is the subject of *rhetoric*, which is often viewed negatively (e.g., as Sophistry), but has a positive contribution as the art of clear communication [93].

A comprehensive treatment of assurance cases must draw on all these areas of inquiry (indeed, one of the standards for assurance cases requires them to be “compelling, comprehensible and valid” [124]). An assurance case surely aims for the truth, so we need to understand something of logic and epistemology. This understanding can save us from major errors, but the ultimate evaluation of an assurance case always rests on human judgment. Unfortunately, human beings are prone to confirmation bias, so it will be wise to subject the case to dialectical scrutiny. But we do not want its scrutineers nor its ultimate consumers to be flummoxed by gratuitous complexity in the case—one of the goals of an assurance case is to *communicate* its argument—so some awareness of rhetorical theory could be useful. Accordingly, we briefly examine each of these areas in the subsections that follow.

5.1.1 Logic

Logic, the study of valid reasoning, is obviously relevant to any treatment of assurance cases. More controversial is the idea that conventional or *deductive* logic is the appropriate model for argumentation in assurance cases; in this model, an assurance case argument should have the character of a proof. An alternative view is that *inductive* logic is the appropriate model for assurance cases. We examine inductive logic in Section 5.2.1; here we examine deductive logic, generally referred to as simply *logic*.

Aristotle was the first to undertake a systematic study of logic and his formulation of the *syllogism* dominated the topic for 2,000 years. A canonical example is shown below.

$$\begin{array}{r@{\ :\ }l} \text{Premise 1} & \text{All men are mortal} \\ \text{Premise 2} & \text{Socrates is a man} \\ \hline \text{Conclusion} & \text{Socrates is mortal} \end{array} \tag{5.1}$$

Modern logic builds on the 19th Century innovations of Boole and Frege, of which the most important were the introduction of explicit quantification with bound variables and functional notation for predicates (e.g., man(x) below).[1] Using these, the example above can be rendered as follows.[2]

$$\begin{array}{r@{\ :\ }l} \text{Premise 1} & \texttt{FORALL}\ x : \text{man}(x)\ \texttt{IMPLIES}\ \text{mortal}(x) \\ \text{Instance 1.1} & \text{man(Socrates)}\ \texttt{IMPLIES}\ \text{mortal(Socrates)} \\ \text{Premise 2} & \text{man(Socrates)} \\ \hline \text{Conclusion} & \text{mortal(Socrates)} \end{array} \tag{5.2}$$

Here, derivation of the line labeled "Instance 1.1" from Premise 1, and derivation of the Conclusion from that instance and Premise 2, follow by rules of logic. In this case, we are using First Order Logic, and the rules involved are called *specialization* and *modus ponens*, respectively.

The point about logic is that it guarantees that if the conclusion can be derived from the premises by applying the rules of logic, then it will be a true statement about the world, provided the premises are. Thus (and this was another key contribution of Aristotle), logic separates (truth of) the subject matter of an argument (encoded in its premises) from validity of the reasoning performed upon it. This separation requires that the reasoning in the argument above must be valid, independently of its subject matter: that is, independently of what "man," "mortal," and "Socrates" mean. We can illustrate this by replacing these specific terms (consistently) by arbitrary symbols as follows.

$$\begin{array}{r@{\ :\ }l} \text{Premise 1} & \texttt{FORALL}\ x : p(x)\ \texttt{IMPLIES}\ q(x) \\ \text{Instance 1.1} & p(a)\ \texttt{IMPLIES}\ q(a) \\ \text{Premise 2} & p(a) \\ \hline \text{Conclusion} & q(a) \end{array} \tag{5.3}$$

[1] Explicit quantification is far more expressive than the syllogism; for example, "every man has a father" requires double quantification:

$$\texttt{FORALL}\ x : \text{man}(x)\ \texttt{IMPLIES}\ \texttt{EXISTS}\ y : \text{man}(y)\ \texttt{AND}\ \text{father}(y, x).$$

[2] There are other ways to do this; in a multi-sorted logic, we could dispense with the predicate "man" and treat this as a sort, but these variations do not affect the substance of the discussion.

Logic focuses on what `FORALL` and `IMPLIES` (and other quantifiers and connectives) mean, and provides rules for these, while p, q, and a are left open to interpretation (e.g., as "man," "mortal," and "Socrates," respectively). A *valid* argument is one that is true in *all interpretations*, meaning that we can substitute anything for p, q, and a, and the conclusion will be true if the premises are. Notice that an argument is valid (or not), independently of whether its premises are true. To draw a useful conclusion, the argument must be *sound*: that is, it must be valid *and* its premises must be true statements about the world. Validity can be established using the laws of the logic concerned (and partially or completely automated using a theorem prover for that logic); the truth of the premises can be established only by human judgement.

The subject matter of our example is the mortality of Socrates, and our knowledge of relevant aspects of the world is recorded in the two premises of (5.2). In logic, these are referred to as "nonlogical axioms" ("nonlogical" because they are not axioms of logic itself, but are about the subject matter of this specific argument) and, as far as logic is concerned, they have equal status: both must be true if the conclusion is to be so.

But as we discussed in Chapter 3, in an assurance case, premises like the one labeled Premise 1 (i.e., having the form of an implication) are representative of the interior or reasoning steps of our argument, while those like the one labeled Premise 2 are representative of its evidential steps. Thus, we validate the first kind of premise by introspection and discussion, and the second kind by the collection and examination of evidence. There is nothing in logic that mandates this approach to the validation of nonlogical axioms, it merely seems appropriate to the kind of arguments that are used in assurance cases.

An argument expressed in logic as in (5.2) actually provides a *proof* of its conclusion. If we take logic and proof as our model for the argument in an assurance case (and, as we noted earlier, this is a controversial position), then there are two things we must do to be sure that a case is sound: check that the premises are true (which itself divides into two parts, with different kinds of checks for the reasoning and the evidential steps), and check that the rules of logic are applied correctly. There are two ways to perform the latter check: rely on human judgment, or use automation.

There is much experimental evidence that humans are not, in general, very good at logical reasoning and fail even quite simple tests (Kahneman gives several examples [74][3]). We might expect, or hope, that those who develop and review assurance cases will perform better than the general population, but the evidence

[3] One is the following sylogism, which many university students (incorrectly) declare to be valid.

Premise 1 : All roses are flowers
Premise 2 : Some flowers fade quickly
Conclusion : Some roses fade quickly

suggests otherwise. Greenwell and colleagues examined three published assurance cases and found that all of them employed fallacious reasoning [47]. These cases did not frame their arguments deductively, so the fallacies that were observed do not exactly correspond to logic errors: indeed, one of the reasons for proposing that assurance cases should be framed deductively is that this imposes a discipline that renders such fallacies less likely and detection more certain.

One might also hope that independent, external review would detect logic errors, and this was indeed how the fallacies reported by Greenwell *et al* were found. However, close reading of their paper [47] shows that different reviewers found different errors, so this might not be a reliable method.

An alternative is to use automated deduction—theorem provers—to check for logic errors. Internally, such an automated check would abstract an assurance case argument to its logical form, for example, (5.1) would be abstracted to (5.3), and the theorem prover would verify its validity or report an error. The speed and effectiveness of automated deduction is related to the expressiveness of the underlying logic. Even the weakest useful logic—propositional logic—is NP Complete, while richer logics such as First Order and Higher Order Logic, especially when enriched with theories such as arithmetic, are highly undecidable. Despite these theoretical barriers, modern theorem provers are extraordinarily powerful and effective on many problems of practical interest. It is routine to apply SAT solvers (for propositional logic) and SMT solvers (SAT enriched with theories such as arithmetic and datatypes) to problems with millions of terms. As we discussed in Section 3.1, the interior steps of an assurance case argument should almost certainly take the form of Horn clauses (recall (3.1) on page 29) and this, a restricted form of First Order Logic, has very effective automation.

In addition to checking that our arguments are valid, we should also check for other kinds of flaw. The most egregious flaw is (sets of) premises that contradict each other. An example (5.4) appears later, where we note that the problem with inconsistent premises is that they can be used to prove *anything*. It is feasible to demonstrate consistency with a theorem prover (e.g., by exhibiting a constructively-defined interpretation), and for simple logics it is easy to detect inconsistency automatically, or to exclude its possibility (e.g., in the absence of explicit negation, Horn clauses cannot be inconsistent). Other simple checks can ensure that the argument is connected, non-circular, and complete.

Automated checks of the kind suggested above will reliably detect logic errors in the argument of an assurance case; for example they would identify the logic error in the following fallacious variant on our original syllogism.

$$\begin{array}{rl} \text{Premise 1}: & \text{All men are mortal}\\ \text{Premise 2}: & \text{Socrates is a man}\\ \hline \text{Conclusion}: & \text{Socrates is dead} \end{array}$$

However, automation cannot detect errors in the subject matter of our arguments; that is, in the premises. To detect the fallacies in the following two arguments, we must examine the content of their premises.

$$\begin{array}{rl}
\text{Premise 1}: & \text{All men are blue-eyed} \\
\text{Premise 2}: & \text{Socrates is a man} \\
\hline
\text{Conclusion}: & \text{Socrates is blue-eyed}
\end{array}$$

In the example above, it is Premise 1 that is suspect: it is representative of a flawed reasoning step in an assurance case. In the example below, it is Premise 2 that is suspect: it represents a flawed evidential step in an assurance case.

$$\begin{array}{rl}
\text{Premise 1}: & \text{All men are mortal} \\
\text{Premise 2}: & \text{Spiderman is a man} \\
\hline
\text{Conclusion}: & \text{Spiderman is mortal}
\end{array}$$

The only way to validate the premises to an assurance case is through human challenge and review. In the next section, we will look at methods for adjusting logic so that it can support some kinds of challenges to the premises of an argument.

5.1.2 Defeasible Logic

Let us return to our original argument about Socrates (5.1) and suppose that a reviewer challenges Premise 1: he says "I have a CD at home called 'The Immortal James Brown,'[4] so Premise 1 can't be right." This new claim about James Brown could be expressed in logic and added to (5.2) labeled as Premise 3, as shown below.

$$\begin{array}{rl}
\text{Premise 1}: & \texttt{FORALL}\ x : \text{man}(x)\ \texttt{IMPLIES}\ \text{mortal}(x) \\
\text{Instance 1.1}: & \text{man(Socrates)}\ \texttt{IMPLIES}\ \text{mortal(Socrates)} \\
\text{Instance 1.2}: & \text{man(JamesBrown)}\ \texttt{IMPLIES}\ \text{mortal(JamesBrown)} \\
\text{Premise 2}: & \text{man(Socrates)} \\
\text{Premise 3}: & \text{man(JamesBrown)}\ \texttt{AND}\ \texttt{NOT}\ \text{mortal(JamesBrown)} \\
\hline
\text{Conclusion}: & ?
\end{array} \tag{5.4}$$

The problem is that this premise contradicts Instance 1.2 of Premise 1. If we interpret this in conventional logic, the contradiction among the premises renders the argument unsound, and we can conclude nothing.[5]

But there are some applications where it is reasonable to proceed with inconsistent premises and to make adjustments to the rules of logic so that they deliver a "reasonable" conclusion. Examples are interpreting information from sensors that

[4] The CD in question is actually called "Immortal R&B Masters: James Brown."

[5] Actually, the problem with contradictory premises is that they can be used to prove *anything*.

periodically update their readings, and resolving arguments on topics where participants hold different views, such as ethics, politics, or aesthetics.

When a new object enters the field of view of a robot, for example, some of its cameras may detect the object earlier than others; later on, one feature detection algorithm may report a person, while another reports multiple people, and a third reports an inanimate source of heat. If the robot uses deduction to plan its actions, then its deductive apparatus must produce useful results from such incomplete or conflicting information and must revise the plan as new information arrives. The same idea would apply to an emergency room physician, updating her diagnosis and adjusting her treatment plan as new observations and test results become available. In arguments on controversial topics, participants might announce general principles that are in conflict, but then qualify them in different ways and possibly reach agreement on particular cases. As in the robot, the logic that interprets their argument must cope with evolving sets of premises that may be incomplete and inconsistent. Similarly, a jury that convicts a defendant in a criminal case must decide "beyond reasonable doubt" in the face of contradictory accounts.

Adjusting conventional logic to cope with these challenges has been studied under different names in different fields. Logic speaks of "nonmonotonic" logics, while artificial intelligence and argumentation speak of "defeasible" logics. One approach is to prefer the most specific rules that apply, so that in (5.4) we can conclude that James Brown is not mortal (preferring Premise 3 over Premise 1 and its Instance 1.2), while Socrates is (provided he and James Brown are not the same).

Of course, a proponent of the original argument (5.2) might observe that James Brown is dead (citing Google) and therefore indubitably mortal. If this were added to the argument, we would have two contradictory premises about James Brown that are equally specific and need some way to choose between them.

A popular approach is to note whether premises *defeat* each other and track whether defeaters are themselves defeated. For example, Premise 3 defeats Instance 1.2, but the new premise about James Brown being dead would defeat Premise 3, thereby restoring Instance 1.2. There is an active area of research with an extensive literature that explores and develops these ideas [17, 88].

A related approach defines an *attack* relation between entire arguments (note that the *defeats* relation is between premises); roughly, one argument attacks another if it has premises that defeat some of those in the other. For example, argument (5.4) attacks the original argument (5.2). The attack relation yields an *argumentation structure*, which is just a graph (i.e., the arguments themselves are abstracted to uninterpreted nodes), and graph theoretic algorithms are defined that calculate the surviving arguments. This is another active area of research with a large literature [17].

These treatments of defeasible reasoning are of great intellectual interest and probably of practical value in fields where it is necessary to draw reasonable conclusions in the face of changing, incomplete, or inconsistent information. But assurance cases, in my opinion, are not like this and, in consequence, I do not see a rôle for defeasible argumentation in evaluation of assurance cases. I appreciate that developers and reviewers of assurance cases are neither infallible nor omniscient, but their responsibility is to do the best they can; if they encounter incomplete, inconsistent, or disputed elements in their reasoning or evidence, they should not use a defeasible logic to calculate a “reasonable” conclusion; instead, they should resolve the problems, record their reasoning, and apply conventional logic to derive a deductively valid conclusion.

Of course, this raises the questions how incomplete, inconsistent, and just plain wrong information or reasoning might be detected in an assurance case argument, how reviewers might actively challenge an argument, and how the consequences of challenges might be calculated. Certainly, the field of defeasible logic provides the very useful notion of “defeater” and it is conceivable that one way to challenge an assurance case argument is to propose defeaters and then use defeasible logic to calculate the consequences—but this would be for the purpose of investigation, not evaluation. We consider these topics in Section 5.2, but note that defeasible logic builds on conventional, deductive logic, so we should first consider the alternative point of view, that an assurance case is an *inductive* argument. The work of Stephen Toulmin provides an influential account of nondeductive arguments and this is the topic of the next section.

5.1.3 Inductive and Toulmin-Style Arguments

We noted in section 5.1.1 that conventional logic focuses on the meaning attached to quantifiers like `FORALL` and logical connectives like `IMPLIES` and provides rules so that arguments employing these can derive valid conclusions no matter what meanings are attached to the other “nonlogical” terms appearing in the argument. Toulmin [122] observes that this privileges a certain kind of reasoning and neglects others that may be more useful and important in the real world. In particular, the deductive `IMPLIES` is about certainty, whereas in daily life we often assert more nuanced relations among terms and qualify implications with words like “probably,” “possibly,” “presumably” and so on, or we say merely “this *suggests* that.” One of Toulmin’s examples, which may be compared to the syllogism (5.1) is the following.

$$\begin{array}{rl} \text{Premise 1}: & \text{Scarcely any Swedes are Roman Catholics} \\ \text{Premise 2}: & \text{Peterson is a Swede} \\ \hline \text{Conclusion}: & \text{Almost certainly, Peterson is not Roman Catholic} \end{array} \tag{5.5}$$

This introduces the idea of an *inductive argument*, in which truth of the premises does not (as in a *deductive* argument) guarantee truth of the conclusion, but supports it with various degrees of force. But once we abandon the framework of deductive logic, we lose its deep metatheory, its algorithms, and its tools. There are approaches that try to reconcile logic with less-than-certain inference, such as probabilistic and Markov logics (which are sketched in Section 5.3), but Toulmin had a different goal. He sought to develop a style of argument that more closely integrated our knowledge of a subject and our reasoning about it—in contrast to deduction, where the logic and the subject matter of an argument are separated.

Toulmin's motivation derived from philosophy and epistemology (i.e., to challenge the primacy of deductive reasoning) but his work was adopted by some of those working in rhetoric, law, and other fields that concern real-world arguments. Those who first developed the ideas of safety cases were conscious that absolute certainty is unattainable, for it is impossible to know everything that might affect the safety of a system, and so they, too, drew on Toulmin's approach.

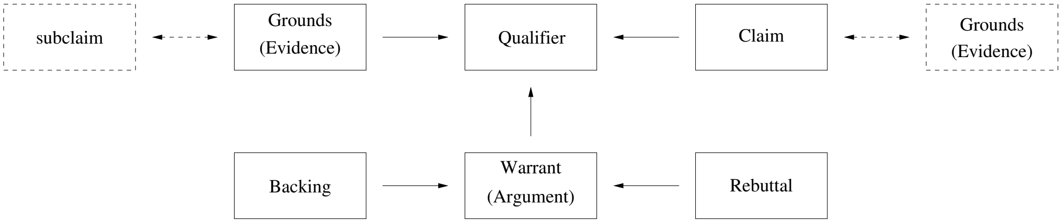


Figure 5.1: Toulmin's Model of Argument

Toulmin's model of argument has the following six elements (the descriptions are taken from [3]), which are also portrayed in Figure 5.1.

Claim: This is the expressed opinion or conclusion that the arguer wants accepted by the audience.

Grounds: This is the evidence or data for the claim.

Qualifier: An adverbial phrase indicating the strength of the claim (e.g., certainly, presumably, probably, possibly, etc.).

Warrant: The reasoning or argument (e.g., rules or principles) for connecting the grounds to the claim.

Backing: Further facts or reasoning used to support or legitimate the warrant.

Rebuttal: Circumstances or conditions that cast doubt on the argument; it represents any reservations or "exceptions to the rule" that undermine the reasoning expressed in the warrant or the backing for it.

Figure 5.1 portrays a single argument step and suggests how these can be chained together, with the (sub)claim at one level providing the grounds (evidence) at a higher level. Toulmin's original formulation did not anticipate this hierarchical form and used a less symmetrical graphical representation; explicit chaining of argument steps seems to be due to Newman and Marshall [85].

If we were to rotate Figure 5.1 through 90 degrees counterclockwise, it would closely resemble Figure 4.1, which portrayed the basic argument step used in CAE notation. This is no accident for, as noted, the developers of CAE and GSN explicitly acknowledge a debt to Toulmin. The *claim*, *grounds*, and *warrant* of Toulmin's approach correspond to the claim, evidence, and justification of an argument step in an assurance case.

Toulmin's *qualifier* seems not to be employed in assurance cases, even in those frameworks such as GSN (and, to a lesser extent, CAE) that explicitly endorse inductive (i.e., less than definitive) reasoning. My opinion is that any assurance case that is not fully deductive should employ qualifiers to indicate for each reasoning step whether that step should be interpreted deductively (i.e., the premises *imply* the conclusion) or inductively (i.e., the premises *suggest* the conclusion).

Toulmin's *backing* seems to function rather like a confidence item in an assurance case: that is, it provides additional reasons why we should believe the claim, while the *rebuttal* is rather like a negated assumption (i.e., it specifies conditions under which the argument step does not apply).

Toulmin's approach is often used in both teaching and performing the analysis of informal arguments: the idea is to identify the claim, warrant, qualifier, and so on and then consider these within Toulmin's framework. Adelman and colleagues conducted an empirical evaluation of such an application of Toulmin's approach in which subjects were asked to evaluate the soundness of arguments presented in two concocted "magazine articles" [3]. The results were inconclusive, and the authors suggest "one needs to be cautious of the claimed value of structured argumentation tools employing the Toulmin formalism." This particular experiment does not seem well-aligned with possible use of Toulmin's approach in assurance cases, but the suggestion of caution is appropriate.

The case that Toulmin advances against classical, deductive logic has some appeal when the topics of discourse are ethics, religion, or aesthetics, say, but it is less persuasive for the topic of assurance (though Cassano and Maibaum provide a sympathetic account [15]). There will certainly be areas of doubt in an assurance case, and human judgment and experience may be the appropriate recourse, but these doubts concern our ignorance or uncertainty about the world (i.e., the formulation of our premises) and do not require reformulation of the notion of logic (i.e., rejection of the idea that validity of the reasoning in an argument can be separated from the truth of its subject matter). This is different than arguments in ethics, for example, where the entire basis for debate may be contested and reasonable people may come to different conclusions.

There are several systems that provide support for representation and evaluation of human arguments using Toulmin's model. *Araucaria* is one that is widely-cited and seems well-supported [96]; it is available at `http://araucaria.computing.dundee.ac.uk/`. Araucaria also supports other styles of human argument and is able to represent arguments using various forms of diagram [97].

5.1.4 Argumentation and Dialectics

Argumentation is an academic field that mainly focuses on the back-and-forth or dialectical nature of debate and argument. One of the goals of dialectical debate is to reach an agreed conclusion, so the methods employed are sometimes referred to as agreement technologies. Since evaluation of assurance cases should probably involve back-and-forth debate with reviewers, tools and methods for argumentation and agreement could be relevant and useful.

Unfortunately, the fields of argumentation and assurance developed in mutual isolation. The journal *Argumentation* (published by Springer) contains no reference to assurance or safety cases among more than 1,000 published articles available online. Neither does a major compendium on agreement technologies [88]. However, some recent work on exploration of assurance cases does draw on work in argumentation. In particular, work by Takai and Kido [119] builds on ideas from the Carneades argumentation framework [44] and, dually, the Argumentation Interchange Format (AIF) standard [16] does mention the ASCE assurance case tool.

Carneades is a system that supports dialectical reasoning, allowing a subargument to be *pro* or *con* its conclusion and allowing weights to be attached to evidence. A *proof standard* is calculated by "adding up" the *pro*s and *con*s supporting the conclusion and their attendant weights. For example, a claim is "in" if it is not the target of a *con* that is itself "in" (unless it is also the target of an "in" *pro*...); a conclusion is supported to the *preponderance of evidence* proof standard if it has at least one *pro* argument that is "in" and weighs more than any "in" *con* argument. The system, which is available at `http://carneades.github.io/`, provides several kinds of argument graphs for visualizing arguments.

We have now surveyed four interpretations for the argumentation employed in assurance cases: classical logic, defeasible reasoning, Toulmin's approach, and dialectics. Classical logic regards an argument as a proof, defeasible reasoning tolerates inconsistency and provides methods for resolving contested arguments, Toulmin's approach is representative of methods that employ inductive and informal logic, while dialectical methods support the to-and-fro of debated arguments.

In the following section we examine the application of these methods to assurance cases.

5.2 Assessing the Soundness and Strength of an Assurance Case

There is uncertainty in the world, and imperfection in human understanding and knowledge about the world—even about artifacts of our own construction. Thus, an assurance case cannot establish conclusively that a system is safe, or has some other desired attribute. Regulatory and certification regimes recognize this and do not expect assurance cases to guarantee that mishaps cannot occur, merely that they will be suitably rare (with tolerable likelihood inversely proportional to severity).

As noted in the previous paragraph, there are two sources of uncertainty in the claim delivered by an assurance case. One is uncertainty *in* the world (e.g., random hardware failures), the other is imperfection in the reasoning and the evidence *about* the world employed by the argument of the case. We cannot do much about the first of these (although we must measure or calculate its statistical properties and design to these constraints) but it is our responsibility to control and evaluate the second. Observe, however, that we do not always expect or desire an assurance case to be fully sound, or even "as sound as possible": in DO-178C, for example, the required objectives are deliberately reduced or weakened as we move from Level A to Level D software, so that even if the implicit assurance case for DO-178C Level A were totally sound, those for the lower levels cannot be—they are deliberately flawed. So the question is: given an assurance case, how can we determine what are its flaws (both deliberate and accidental), and how can we estimate their impact on our confidence in the case, and on the (probabilistic) safety of the system?

5.2.1 What Assurance Case Assessment Must Accomplish

It follows from the considerations above that the overall argument of an assurance case will be inductive rather than deductive: that is, it strongly suggests, but cannot prove, that its top-level claim is true. But does this mean that every step of the argument may likewise be inductive? To examine this question, we need to recall the structure of an assurance case argument.

In Section 3.1, we proposed that arguments should be presented in "simple form" and noted that any argument can easily be converted to this form (recall Figure 3.2). In simple form, all argument steps are either *reasoning* (or interior) steps, or *evidential* (or leaf) steps; the former establish a claim from subclaims (which themselves must be established by lower-level arguments), while the latter establish subclaims from evidence. Evidential steps are necessarily inductive (we explain why this is so below), but reasoning steps could be either inductive or deductive.

If a reasoning step is inductive, it means that we have some doubt whether the subclaims are sufficient to establish the claim. But how much doubt is permissible? And can we estimate its magnitude?

We may surely assume that any inductive step is "almost" deductive. That is to say, the following generic inductive step

$$p_1 \texttt{ AND } p_2 \texttt{ AND } \cdots \texttt{ AND } p_n \texttt{ SUGGESTS } c \tag{5.6}$$

would become deductive if we added some missing (and presumably unknown) subclaim or assumption a (which, of course, may actually be a conjunction of smaller subclaims), as shown below.[6] (It may be necessary to adjust the existing subclaims p_1 to p'_1 and so on if, for example, the originals are inconsistent with a).

$$a \texttt{ AND } p'_1 \texttt{ AND } p'_2 \texttt{ AND } \cdots \texttt{ AND } p'_n \texttt{ IMPLIES } c. \tag{5.7}$$

If we cannot imagine such a "repair," then surely (5.6) must be utterly fallacious. It then seems that any estimation of the doubt in an inductive step like (5.6) must concern the "gap" represented by a. Now, if we knew anything at all about a it would be irresponsible not to add it to the argument. But since we did not do so, we must be ignorant of a and it follows that we cannot estimate the doubt in inductive argument steps.

If we cannot estimate the magnitude of our doubt, can we at least reduce it? This seems to be the purpose of confidence claims, but what exactly is their logical rôle? One possibility is that confidence claims eliminate some sources of doubt. For example, we may doubt that the subclaims imply the claim *in general*, but the confidence claims restrict the circumstances so that the implication is true *in this case*. But such use of confidence claims amounts to a "repair" in the sense used above: these claims are really assumptions that should be added to the argument as additional subclaims (recall the derivation following (3.2) on page 38), thereby making it deductively sound, or at least less inductive.

The logical rôle of other kinds of confidence claims is less clear; my suspicion is that they serve no purpose at all. In my opinion, use of inductive argument steps and confidence claims opens Pandora's Box, for these cannot be evaluated in any strict sense and there is no way to determine that we have "enough." There is then a temptation to employ complex, but still inductive, reasoning steps, buttressed with numerous confidence claims "just in case." As Hawkins *et al* observe, this results in "voluminous, rambling, ad infinitum arguments" [55]. They recommend an approach in which confidence claims are removed from the main safety argument but linked to it through assurance claim points (ACPs) at restricted locations. These "assured safety arguments" were described in Section 4.2.2. One kind of ACP attaches confidence claims to reasoning steps (specifically, to GSN strategies, yielding "asserted inferences") and therefore seems to address our current topic. However,

[6] There are other, logically equivalent, ways to interpret the repair: for example, we could suppose that the premises stay the same but the conclusion is weakened to the claim $c \texttt{ OR NOT } a$.

the purpose of assured safety arguments is simply to improve the readability of arguments that use confidence claims and [55] provides no guidance on how to assess their contribution. Thus, we know of no established or proposed method to assess the contribution of confidence claims to the evaluation of inductive reasoning steps in assurance arguments.

My recommendation is that the reasoning steps of a assurance argument should be deductive: if the subclaims to the step are true, then truth of the claim must follow. Confidence claims add nothing to deductive reasoning steps and can be eliminated, thereby simplifying formulation of the argument. The main reason to prefer deductive to inductive reasoning steps is that it is clear what their evaluation must accomplish: evaluation of a deductive step must review its content and justification and assent (or not) to the proposition that the subclaims truly imply the claim. A secondary reason is that superfluous subclaims are likely to complicate rather than strengthen the justification for a deductive step. Hence, the requirement for deductive soundness encourages the formulation of precise subclaims and concise arguments.

Recall from Sections 2.3 and 2.4 that early safety cases employed a "narrative" argument from evidence to top claim that was often difficult to comprehend and to evaluate. Structured cases introduced explicit stepwise arguments and graphical representations that facilitated comprehension of the overall case. But cases are generally very large and cannot truly be comprehended *in toto*: a modular or compositional method is required. Observe that soundness of deductive reasoning steps can be assessed in just such a modular fashion, one step at a time. In addition to local soundness, we need also to be sure that subclaims are interpreted consistently between the steps that establish them and the steps that use them, but this, too, is a modular process. Thus, deductive reasoning steps thus support modular assessment in a way that inductive steps do not—for when a step is labeled inductive, we are admitting a gap in our reasoning: we must surely believe either that the gap is insignificant, in which case we could have labeled the step deductive, or that it is taken care of elsewhere, in which case the reasoning is not modular.

The obvious objection to the recommendation for deductive reasoning steps is that it may be very difficult to construct these, and still harder to assemble them into a complete case. One response is that this may accurately reflect the true difficulty of our enterprise, so that simplifications achieved through inductive reasoning will be illusory. Also note that, while the top claim and some of the evidential subclaims may be fixed (by regulation and by available evidence, respectively), we are free to choose the others. Just as formulation of good lemmas can simplify a mathematical proof, so skillful formulation of subclaims may make a deductive assurance argument tractable.

Another potential objection derives from claims that science itself may not support deductive theories. This is a contoversial topic in the philosophy of science that concerns "provisos" (sometime spelled "provisoes") or *ceteris paribus* clauses[7] in statements of scientific laws. For example, we might formulate the law of thermal expansion as follows: "the change in length of a metal bar is directly proportional to the change in temperature." But this is true only if the bar is not partially encased in some unyielding material, and only if no one is hammering the bar flat at one end, and .... This list of provisos is indefinite, so the simple statement of the law (or even a statement with some finite set of provisos) can only be inductively true. Hempel [58] asserts there is a real issue here concerning the way we understand scientific theories and, importantly, the way we attempt to confirm or refute them. Others disagree: in an otherwise sympathetic account of Hempel's work in this area, his student Suppe describes "where Hempel went wrong" [118, pp. 203, 204], and Earman and colleagues outright reject it [35].

Rendered in terms of assurance cases, the issue is the following. During development of an assurance case argument, we may employ a reasoning step asserting that its claim follows from some conjunction of subclaims. The assertion may not be true in general, so we restrict it with additional subclaims representing necessary assumptions and provisos that are true (as other parts of the argument must show) in the context of this particular system. The "proviso problem" is then: how do we know that we have not overlooked some necessary assumption or proviso? The answer is that the justification for the step should explain this, and we may provide evidential subclaims that cite the methods used (e.g., to show that we have not overlooked a hazard, we will supply evidence about the method of hazard analysis employed) but, lacking omniscience, we cannot be totally certain that some "unknown unknown" does not jeopardize the argument. It can then be argued that the only philosophically sound position is to regard the reasoning step (indeed, all reasoning steps) as inductive.

A counterargument would ask: what does this gain us? If we cannot estimate the size and significance of these "unknown unknowns" (and, indeed, we cannot do this, as explained a few pages back in the paragraph following (5.7)) then we may gain philosophical purity but no actionable insight, and we lose the benefits of deductive reasoning. Furthermore, there is also a loss in our psychological position: with an inductive reasoning step we are saying "this claim holds under these provisos, but there may be others," whereas for a deductive step we are saying "this claim holds under these provisos, and this is where we make our stand." This alerts our reviewers and raises the stakes on our justification. There may, of course, be unknown provisos, but we are confident that we have identified the only ones that matter. Philosophically, this is hubris, but for assurance it is the kind of explicit and definitive assertion that induces effective review.

[7] This latin phrase is usually translated "other things being equal."

If, as I propose, all reasoning steps are deductive, yet the overall assurance case is necessarily inductive, it follows that our doubts must focus on the evidential steps. This is exactly as it should be: evidential steps connect the real world (evidence) to the world of concepts (subclaims); these are different kinds of thing and we have only imperfect knowledge of the first (cf. Popper's "Three Worlds" and its interpretation for engineering [115, 116]).

If evidential steps are inductive, it might seem that their evaluation would be susceptible to the same criticisms as inductive reasoning steps. However, I do not think this is so, for the reason that evidential steps are not interpreted the same way as reasoning steps. When an evidential step uses two or more items of evidence to support a subclaim (as, for example, at the lower left of either argument in Figure 3.2), the interpretation is not that the conjunction of the evidence logically supports the subclaim, but that each supports it to some degree and together they support it to a greater degree. The reason we have several items of evidence supporting a single claim is that there are rather few claims that are directly observable. A simple property like length can be measured directly, but more abstract concepts like "correctness" can (in practice) only be inferred from indirect observations; because the observations are indirect and imprecise we combine several of them, in the belief that, together, their different "views" provide an accurate representation of that which cannot be observed directly. Thus, the assessment of evidential steps is not a problem in logic (i.e., we are not *deducing* the claim from the evidence) but in epistemology: we need to assess the extent to which the evidence allows us to *know* the truth of the subclaim. One way to frame these questions of "degree" and "extent" of knowledge and belief is in terms of probabilities.

Bayesian Confirmation Theory [34] (a subfield of Bayesian Epistemology [13]) holds that evidence E supports or confirms a hypothesis (here, a subclaim) H if the probability of H given E is greater than the prior probability of H alone: $P(H \mid E) > P(H)$. This is a qualitative measure: it does not indicate how strongly E supports H. There is no agreement in the literature on the best quantitative measure: the simplest is $P(H \mid E) - P(H)$, Fitelson [42] makes a good case for $\log \frac{P(E \mid H)}{P(E \mid \neg H)}$, while Joyce [72] argues that different measures serve different purposes. Confirmation theory can produce surprising results. One is the "Paradox of the Ravens" where evidence of a non-black non-raven (e.g., a white shoe) provides confirmation (albeit weak) for the claim "all ravens are black." Another is that evidence may confirm a conjunction of claims $p \wedge q$ more strongly than either p or q alone, even though $p \wedge q$ is always less probable than p or q considered alone [90].[8] The Paradox of the Ravens is sensitive to the confirmation measure used whereas the second result is not—rather, it highlights the difference between confirmation and probability.

[8] The reason is that the new evidence produces a bigger "bump" in the probability of $p \wedge q$ than in those of p or q; an example is the probability that two children are girls, given the new evidence that they are identical twins.

Bayesian Confirmation theory is generally proposed as a model for the evaluation of scientific theories, where we have some background knowledge or beliefs and are considering the impact of a new item of evidence (e.g., an observation or experiment). In an assurance case, we typically profess prior ignorance and have multiple items of evidence $E_1, E_2, \ldots E_n$ that need to be combined in some way before applying the confirmation measure. Prior ignorance can be modeled in different ways (e.g., we might suppose the prior probability that the subclaim holds is 0, or 0.5, or is uniformly distributed) but it will presumably be treated the same across all cases and so the distinction between confirmation and probability vanishes and we merely need a way to assess $P(H \mid E_1 \wedge E_2 \wedge \ldots \wedge E_n)$. I sketch an approach using Bayesian Belief Networks (BBNs) in Section 5.3. Alternatives to BBNs include "possibility theory" based on fuzzy sets and the Dempster-Shafer "theory of evidence" (all three are compared in [132]).

Whether we use these formal approaches to confirmation and quantify our confidence, or rely on more informal approaches, the essential point is that the evidential steps of an assurance argument are assessed by "weighing" the evidence using ideas that can be grounded in probability and epistemology. When the weight of evidence supplied in an evidential step crosses some threshold, we regard its claim as a "settled fact" that may then be propagated through the reasoning steps in the upper levels of the argument.

A reasonable question then is: why can we not apply the same ideas to reasoning steps and thereby allow these to be inductive? Some advocate doing this; for example [7, 136] apply Dempster-Shafer analysis throughout an assurance case, while [28] uses BBNs in a similar way. In my opinion we should not do this because reasoning steps are about just that—reasoning—not epistemology, and should be evaluated by logical rather than epistemological methods. Epistemological methods are insensitive to the logical content of reasoning steps, so analyzing the whole argument by epistemological methods is effectively to flatten the argument by removing subclaims so that only evidence is left, as portrayed in Figure 5.2. But this just takes us back to approaches such as DO-178C, where all we have is a collection of evidence, and loses the whole essence of argument-based assurance.

The idea of structured argument is to facilitate modular comprehension and assessment of the case. If we look at this top-down, we divide each claim into components whose conjunction implies the claim, and recurse down to subclaims supported by evidence; if we look at it bottom-up, we treat each evidentially-suported subclaim as an independently settled fact and conjoin these to produce higher-level subclaims that combine recursively to deliver the top claim. In either case (and, of course, a real assurance case is developed and understood through a combination of top-down, bottom-up, and middle-out reasoning), the interior or reasoning steps are understood logically—as conjunctions—of recursively established facts, and not as accumulations of evidence. And since we are free to choose the reasoning steps

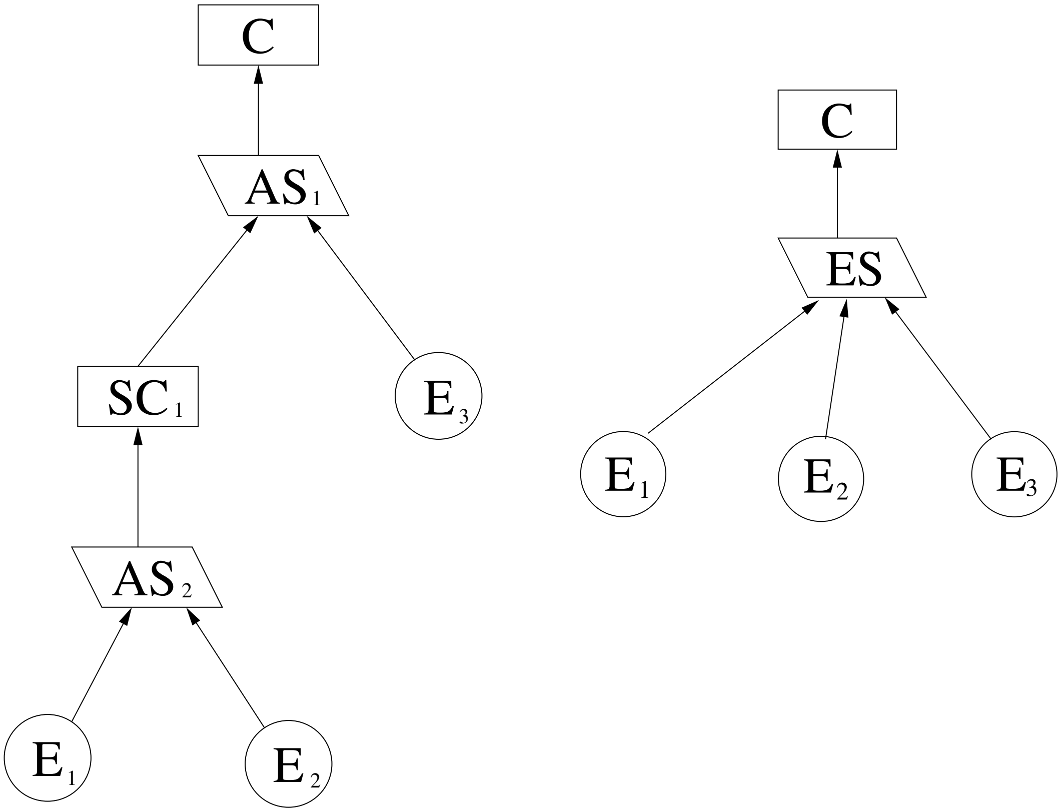


Figure 5.2: Flattening an Argument by Eliminating Subclaims

and their subclaims, we can choose to ensure that each step is deductive: if the subclaims to a reasoning step do not imply its claim, we should have chosen a different claim or subclaims.

It could be argued that subclaims that are strong enough to imply our claim might themselves have poor evidential support: we would be better off, this argument might go, to use subclaims that have stronger evidential support, even if these provide merely inductive support for the desired claim. My response is that this is a false dilemma: neither approach is satisfactory and if we cannot build a better assurance argument for the system as designed, we should change its design.

Let us now draw together the various threads of the discussion above and summarize our recommendations. First, arguments should be normalized to simple form, where we can distinguish reasoning steps from evidential steps and apply different methods to each. Next, evidential steps are evaluated epistemologically by weighing the evidence and its supporting justifications to ensure that collectively these cross some threshold that allows the evidential subclaim to be regarded as a settled fact.

Reasoning steps are evaluated in logic to ensure that the conjunction of subclaims deductively entails the claim. Finally, when an argument is logically valid and all its steps satisfy these criteria, we will say that the argument is *sound*.

In the next section we consider how these evaluations might be performed, and then examine the case of "graduated assurance" (as in DO-178C) where the argument is deliberately weakened for systems that pose lesser risks. This will raise the topic of the *strength* of a (sound) argument and segue to quantification of argument strength in terms of probabilities.

5.2.2 How Assurance Case Assessment Might Be Performed

The overall evaluation of an assurance case argument is accomplished by determining that all its evidential steps cross some threshold of epistemic credibility, which may be either a qualitative or (e.g., if BBNs are employed) a quantitative judgment, and that all its reasoning steps are logically true (either inductively or, preferably, deductively) in their intended interpretation.

It requires human judgment to make these determinations. Human judgment is an excellent instrument, but it is known to be prone to confirmation bias (i.e., a tendency to seek information that will confirm a hypothesis, rather than refute it) so there is legitimate concern that developers of an assurance case will seek to justify it rather than seriously challenge it [79, 127]. On the other hand, motivated outsiders are known to be effective critics of the arguments of others. Evolutionary psychology offers explanations for these phenomena [84].

So now we should ask whether there are systematic procedures or techniques that can reduce or compensate for confirmation bias on the part of assurance case developers, or increase the effectiveness of external reviewers.

First, we should observe that one notorious assurance case—that for the UK Royal Air Force Nimrod airplane, one of which exploded in mid-air with the loss of all onboard—was egregiously flawed [49]. Henceforth, we assume that assurance cases are developed in a context with sufficient management discipline and control on the custody of evidence that this level of malpractice (essentially, failure to disclose known hazards) is prevented.

Next, it surely desirable that assurance cases should be aided by some degree of computerized support. At the very least, some method for managing and cross-linking claims, evidence, arguments, justifications, and other components of a case should be provided, together with automation to ensure their consistent use and to report on the overall coherence and completeness of the assurance case. This level of support simply organizes and manages text and files and is largely independent of the semantics ascribed to the case. Graphical presentation (and, if desired, a GUI interface) can be provided, too.

Once we have this level of computer support, it becomes feasible to explore and probe a case systematically. Graphical representations may be useful for exploration and navigation of a case, but I am skeptical that they can provide much help in the assessment of a case. For credible assessment, we need to examine the content of the case in detail. First, we should review, discuss, and revise as necessary, the narrative justification for each argument step and the meanings (i.e., intended interpretations) attached to its claim and each of its subclaims or items of evidence. Then, in the same way that we look for hazards when exploring the safety of a design, so we can look for *defeaters* to the argument. (What we are referring to as "defeaters" are generally called "Assurance Deficits" in the GSN community.)

When the reasoning steps of an argument are deductive, a defeater for a step is any reason for doubting that its claim must be true when its subclaims are. One approach is to hypothesize a situation that contradicts the claim, then check that it is excluded by at least one of the subclaims. For example, in the `DO-178C HLR Verification Process` argument of 16 on page 46, we might hypothesize that the HLR could specify a behavior about which the SR says nothing (potentially an unintended function), thereby contradicting the claim `HLR correctly represent SR`. This is not excluded by the subclaim `HLR comply with SR`, which requires only that behavior that *is* specified in the SR is correctly represented in the HLR. So now it comes down to interpretation of the subclaim `HLR are traceable to SR`. If "traceable" means only that all the explicit conditions governing behavior in the HLR must correctly trace back to conditions in the SR, then we might ask about actions specified in the HLR under `ELSE` or other default clauses (the specification language used for the HLR might provide such clauses while that used for the SR does not). This may lead us to sharpen the interpretation attached to the subclaim `HLR are traceable to SR` and to reexamine the subargument used to establish it.

In general, we propose that the process of challenging the reasoning steps of an assurance case argument can be systematized as a process of hypothesizing potential defeaters. Just as systematic methods have emerged for undertaking hazard analysis on systems, so might systematic methods be developed to generate substantive defeaters to assurance arguments. Analysis of these should provide a credible means for assessing the soundness of the reasoning steps of an assurance argument.

Notice that if reasoning steps are allowed to be inductive rather than deductive, the process of postulating and analyzing defeaters is likely to lead to "repair" in the way described for (5.6) on page 89. That is, "`NOT` a" would be proposed as a defeater, found to be effective, and the argument would be revised to yield the deductive step (5.7) with a as an assumption. Thus, even if inductive reasoning steps are considered acceptable, a strenuous challenge process is likely to refine them until they become deductive.

But what if a reasoning step remains inductive even after such scrutiny? Presumably this means we are unable to assent that the subclaims truly imply the claim, but we cannot formulate additional assumptions, constraints, or other explicit caveats that eliminate (or even identify) the source of doubt. I know of no rigorous way to evaluate the "size" of the doubt in such cases, nor do I know how to evaluate the contribution of any "confidence" claims, if the step is interpreted in logic. One possibility is instead to interpret the step epistemically, as discussed below for evidential steps.

Evidential steps are necessarily inductive, but they are assessed epistemically rather than logically and so the process of challenge by potential defeaters works out differently. Recall that an evidential step is assessed by "weighing" the combination of evidence supplied to ensure that it crosses some threshold that allows its claim to be regarded as a "settled fact." A defeater for such a process could hypothesize a case where an artifact violates the claim yet "sneaks though" the evidential checks. Consider, for example, an evidential step that uses testing as evidence to discharge the subclaim that executable object code (EOC) is correct with respect to low level requirements (LLR). Defeaters for this step might challenge the test coverage metric used, how well it is measured, and the quality of the test oracle employed. Responses to these challenges might be careful justification for the choices made and methods employed, or they might add new items of evidence (e.g., independent evaluation of the test oracle). Other defeaters might hypothesize that the EOC fails only on scenarios much longer than those used to establish test coverage. A response might be to offer formal verification (which can cover all possible scenarios) as additional evidence but this might not be considered cost-effective in the given circumstances. An alternative would be to offer sound static analysis, which also covers all possible scenarios, but only for predefined properties (typically runtime exceptions), not the application-specific properties described in the LLR. A third response might argue that the software concerned is a cyclic control function that does not accumulate state and therefore long scenarios are no more revealing than those used to establish coverage.

Whereas assessment of reasoning steps has to determine if they are logically true, assessment of evidential steps has to determine if the evidence is suitably persuasive. The purpose of proposing defeaters for evidential steps is to ensure that "persuasive" is not taken lightly. Assessment of evidential steps and evaluation of the impact of potential defeaters may be by review and debate, but it can also employ probabilistic reasoning, as described later.

In summary, we propose a two-part process for the assessment of assurance case arguments: one part uses epistemic (potentially, probabilistic) methods to evaluate the credibility of evidential steps, the other uses logic to evaluate the truth of deductive reasoning steps. This combination of logical and epistemic/probabilistic methods seems natural and straightforward, but I have not seen it explicitly described elsewhere. Haley and colleagues [50] describe a method where reasoning steps are evaluated in formal logic (they call this the *Outer Argument*) while evidential steps are evaluated informally using Toulmin's approach (they call this the *Inner Argument*). They acknowledge that the inner argument concerns "claims about the world" [50, pp. 140] but, unlike us, they use informal reasoning rather than explicitly epistemic methods.

A consequence of our proposed approach to assessment, as well as our philosophical preference, is that reasoning steps should be deductive. There seems to be no good way to evaluate inductive reasoning steps in logic; one possible response is to collapse all the reasoning below any inductive steps and apply epistemic methods to the combination of evidence at the leaves. But this largely vitiates the purpose of structured argumentation and leaves us with evidence alone.

Evaluation of both reasoning and evidential steps should actively challenge the case by hypothesizing potential defeaters. The idea of defeaters comes from epistemology, where Pollock [91, page 40] defines a *rebutting defeater* as one that (in our terminology) contradicts the claim of an argument step, while an *undercutting defeater* doubts the step itself (i.e., doubts that the claim really does follow from the proffered subclaims or evidence); others subsequently defined *undermining defeaters* as those that doubt some of the evidence or subclaims used in an argument step. The last of these seem inappropriate to the evaluation of assurance arguments: in each step, we provisionally accept the subclaims as "settled facts" and our task is to satisfy ourselves that the claim necessarily follows; evaluation of the subclaims is performed (recursively) in steps where they are the conclusion. Although the general idea of defeaters is valuable, in my opinion this classification of them adds little to the process of human challenge and review for assurance case arguments. On the other hand, it is conceivable that it could be exploited by some form of automated exploration, possibly using the techniques of defeasible reasoning and argumentation outlined in Section 5.1.2.

Closely related to the idea of defeaters is the far older notion of *eliminative induction*, which is a formulation of scientific method due to Francis Bacon (in the *Novum Organum* of 1620). Roughly, this formulation holds that a scientific theory becomes increasingly tenable as reasons for doubting it are eliminated. Perhaps surprisingly, there is a strong connection between this ancient idea and modern Bayesian Confirmation Theory [56].[9]

Goodenough and colleagues advocate eliminative induction both as a way to evaluate assurance cases [129] and as an alternative way to argue confidence in system properties [43]. In both cases, they argue for systematic exploration of defeaters, using the classification mentioned above (rebutting, undercutting, and undermining) to guide a systematic search for reasons why a claim might be falsified or an argument step defeated.

[9] Despite its ancient roots, Eliminative Induction remains a plausible account for how science is *done*; Popperian Falsification [92] explains what science *is*.

In the earlier reference [129], the number of challenges by potential defeaters that an assurance argument successfully withstands is used as a rough measure of confidence in the argument; more specifically, the ratio of the number of defeaters successfully withstood to total number considered is used a measure and referred to as Baconian probability.

Although challenge by potential defeaters seems an excellent way to validate an assurance case, there are objections to Baconian probability as a measure. First, it is not a probability in the mathematical sense (i.e., it does not satisfy the properties expected of a probability, namely the Kolmogorov Axioms); second, it is easy to inflate the measure by proposing numerous similar and easily countered defeaters.

Baconian probability was extensively developed and advocated by Cohen in the 1970s as an alternative to traditional "Pascalian" probability [18] (the book is hard to obtain, a review by Wagner provides a useful summary [126]). His position was hotly disputed—see, for example, the intemperate exchanges between him and Kahneman and Tversky [19, 20, 75]. Although the foundations of subjective probability and Bayesian methods were developed by Ramsey, di Finetti, and others in the 1920s, and 30s, and their application to the "weighing of evidence" was established by Good and others in the 1940s and 50s, Cohen makes no mention of these approaches and, indeed, they were not well known at that time and did not become mainstream until the 1980s and 90s. In my opinion, Bayesian methods provide a more secure foundation for the evaluation of assurance cases than Baconian probabilities. That said, I am aware of no Bayesian proposal for deriving a measure for confidence in an assurance case that accounts for defeaters. We do sketch an approach to the "weighing" of evidence based on BBNs in Section 5.3.1, but this considers only "positive" contributions and not defeaters.

In the second reference [43], Goodenough and colleagues go further and propose that eliminative argumentation should be used directly as a means to argue confidence in system properties. They propose *Confidence Maps* as a graphical representation for the conduct of eliminative induction that is like a "dual" to GSN. Whereas GSN, or any other representation for an assurance case, describes the "positive" argument for safety (or other assured property), a confidence map portrays ways in which it could fail to achieve the desired claim (i.e., its hazards and defeaters) and explains how these are countered. This resonates with a recommendation by Haddon-Cave, who conducted the enquiry into the Nimrod disaster. He found that safety cases were often "compliance-only exercises" that "failed to see the wood for the trees" and provided only "paper safety"; he recommended they should be made more "proportionate and relevant" and renamed "Risk Cases" [49, Chapter 22].

Again, although I am sympathetic to use of defeaters in validation of assurance case arguments, I am less enthusiastic about making defeaters the main focus. A defeater does not stand alone: there must be a positive case for it to "defeat"; furthermore, the most telling defeaters are likely to be those that ask probing questions about how the system works (to achieve safety or other top claim) and these simply cannot be formulated in the absence of a detailed positive argument about how it is meant to work. Also, there is no criterion for believing that a collection of defeaters is in any sense "complete," whereas a positive assurance argument must possess an internal coherence that does entail some notion of completeness.

In summary, my opinion is that assessment of an assurance case requires human judgement to ensure that all its evidential steps provide sufficient "weight" of evidence that their subclaims may be considered "settled facts," and that all its reasoning steps are inductively or (preferably) deductively true: that is, their subclaims truly suggest or (preferably) imply their conclusions. Despite best efforts and good intentions, however, there is a risk of confirmation bias, so the case should be challenged by considering potential defeaters to the argument. In addition to representing the argument in a form that can be readily examined and probed, any tool support for assurance case argumentation should provide a way to document the defeaters that have been considered, and other challenges to which an argument has been exposed, together with their associated analyses. It is a topic for future research to discover whether automated support (perhaps in the form of defeasible reasoning) can assist in exploring assurance case arguments and their defeaters, and whether metrics (such as Baconian "probabilities") or representations (such as confidence maps) add value to the exploration of defeaters.

Related to the soundness of an assurance case argument is the additional idea of its "strength." This topic is significant for assurance regimes that allow reduced levels of assurance for those systems that pose lesser degrees of risk, and is considered in the next section.

5.2.3 Graduated Assurance

DO-178C recognizes that software deployed for different functions may pose different levels of risk and it accepts reduced assurance for airplane software that poses less risk. In particular, the number of assurance objectives are reduced from 71 for Level A software (that with the potential for a "catastrophic" failure condition) to 69 for Level B, 62 for Level C, and 26 for Level D, and the number of objectives that must be performed "with independence" is likewise reduced from 33 to 21, 8, and 5, respectively. This is an example of *graduated assurance*, and it is found in similar form in many standards and guidelines.

On the one hand, this seems very reasonable, but on the other it poses a serious challenge to the idea that an assurance case argument should be sound. We may suppose that the Level A argument is sound, but how can the lower levels be so when they deliberately remove or weaken some of the supporting evidence and, presumably, the implicit argument associated with them?

There seem to be three ways in which an explicit assurance case argument can be weakened in support of graduated assurance, and we consider these in turn.

First, we could simply eliminate certain subclaims or, equivalently, provide nugatory evidence for them (recall 14 on page 42). This surely renders the full argument unsound: any deductive reasoning step that employs the eliminated or trivialized subclaim cannot remain deductively sound with one of its premises removed (unless there is redundancy among them, in which case the original argument should be simplified). This approach reduces a deductively sound argument to one that is, at best, inductive, and possibly unsound; consequently, we deprecate this approach.

Second, we could eliminate or weaken some of the evidence supplied in support of selected subclaims. This is equivalent to "lowering the bar" on what constitutes a "settled fact" and does not threaten the soundness of the argument, but does seem to reduce its strength. Intuitively, the *strength* of a sound assurance case argument is a measure of its evidential support—that is the height of the bars that determine settled facts. If all evidential steps are equally important, then overall strength would seem to be a "weakest link" property—that is, determined by the evidential step with the lowest bar. On the other hand, some claims may be considered more important than others, and the evidential support for more important claims might be given greater weight. Thus, comprehending the consequences for the strength of the overall argument due to weakening individual items of evidence seems to require considerable human skill and experience. It would be very useful to find ways to quantify the "strength" of a sound argument so that the consequences of weakened evidence could be assessed more systematically. We sketch a possible approach in the sections that follow but the results are modest. Nonetheless, they reinforce the idea that weakening should be done "evenly." That is, we should not reduce the evidential support for some subclaims much more than others because overall argument strength does seem to be a weakest link phenomenon (however, we may prefer to leave some evidence at "full strength" because a weakened version is hard to justify).

It could be argued that there is surely no difference between lowering the threshold for evidential support of a given claim and using that same evidence to provide strong support for a weaker claim, which could then provide only inductive support to those reasoning steps that use it—yet I assert that the former is acceptable and the latter is not. My justification is pragmatic: we have a rigorous procedure for evaluating deductive reasoning steps but not inductive ones.

Third, we could restructure the argument. Michael Holloway's reconstruction of the argument implicit in DO-178C [62,63] suggests that this underpins the changes from Level C to Level D of DO-178C. The Low Level Requirements (LLR) and all their attendant objectives are eliminated in going from Level C to Level D; the overall strategy of the argument, based on showing correctness of the EOC with respect to the SR, remains the same, but now employs a single step from HLR to source code without the LLR to provide an intermediate bridge. This also seems a valid form of weakening. It is a topic for future research to discover whether suitable restructurings can be given a concise description that allows their soundness to be derived by calculation from their parent argument.

5.3 Probabilistic Assessment of Assurance Cases

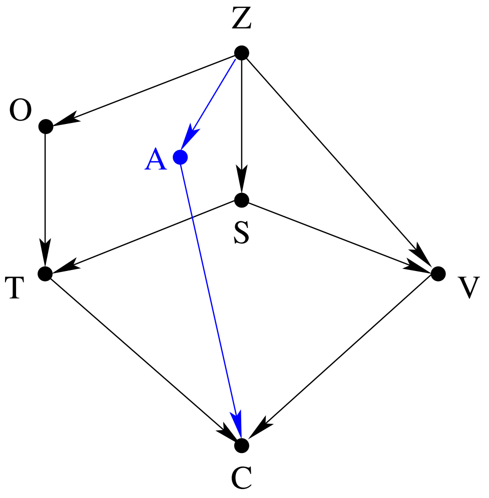
The previous section described how evaluation of deductive reasoning steps is a problem in logic (do the subclaims truly imply the claim?), whereas evaluation of evidential steps is a problem in epistemology (does the evidence amount to knowledge that the claim is true?). We suggested that ideas from Bayesian confirmation theory provide one way to assess the strength of evidential steps. We now illustrate how this might be accomplished using Bayesian Belief Nets (BBNs).

5.3.1 Probabilistic Assessment of Evidential Steps

When we have just a single item of evidence E supporting a claim C, we might use some relation between $P(C \mid E)$ and $P(C)$ as a measure of the extent to which E supports C. These are subjective probabilities [70] that indicate the strength of our posterior and prior beliefs in the claim C.

The task is more complicated when we have multiple items of evidence (this is sometimes called a "multi-legged" assurance step [11]) because these might not be independent and our probabilistic calculation must account for this. Bayes' Theorem is the principal tool for analyzing conditional subjective probabilities: it allows a prior assessment of probability to be updated by new evidence to yield a rational posterior probability. It is difficult to calculate over large numbers of complex conditional (i.e., interdependent) probabilities, but usually the dependencies are relatively sparse and can conveniently be represented by a graph (or network, or "net"—the term used in BBNs) in which arcs indicate dependencies. An example, taken from [80], is shown in Figure 5.3 (ignore, for the time being, the parts associated with A and shown in blue). This represents a "two-legged" argument step in which evidence from testing is combined with that from formal verification. The nodes of the graph represent judgments about components of the argument and the arcs indicate dependencies between these.

The nodes of the graph actually represent random variables but we can most easily understand the construction of the graph by first considering the artifacts from which these are derived. Here, Z is the system specification; from this are derived the actual system S and the test oracle O. Tests T are dependent on both the oracle and the system, while formal verification V depends on the system and its specification. The conclusion or claim C is based on both the test results and the formal verification.



Z: System Specification

A: Assessment of spec'n quality

O: Test Oracle

S: System's true quality

T: Test results

V: Verification outcome

C: Conclusion

Figure 5.3: Example BBN

More precisely, Z is a variable that represents correctness of the system specification: it has two possible values: `correct` (i.e., it accurately represents the true requirements on the system) or `incorrect`. The assessor must attach some prior probability distribution to these possibilities (e.g., 99% confidence it is `correct`, vs. 1% that it is `incorrect`).

O is a variable that represents correctness of the test oracle; it is derived in some way from the specification Z and its probability distribution will be some function of the distribution over the correctness of Z (e.g., if Z is `correct`, we might suppose it is 95% probable that O is `correct`, but if Z is `incorrect`, then it is only 2% probable that O is `correct`).

S is a variable that represents the true (but unknown) quality of the system, stated as a probability of failure on demand. This probability depends on Z: we might suppose that it is 0.99 if Z is `correct`, but only 0.5 if it is `incorrect`.

T is a Boolean variable that represents the outcome of testing. It depends on both the oracle O and the true quality of the system S. Its probability distribution over these is represented by a joint probability table such as the following, which gives the probabilities that the test outcome is judged successful.

Correct System		Incorrect System	
Correct Oracle	Bad Oracle	Correct Oracle	Bad Oracle
100%	50%	5%	30%

V is a Boolean variable that represents a successful verification outcome and it will be defined by joint probability table similar to that for testing.

Finally, C represents the claim that the software is correct, which will be supported only if the system passes verification and no failures are discovered in testing.

In this example, only T and V are directly observable, and C is fully determined by these. Using a BBN tool such as HUGIN Expert [65], it is possible to conduct "what if" exercises on this example to see how prior estimates for the conditional probability distributions of the various nodes are updated by the evidence (i.e., that verification passes and that testing finds no errors) and thereby to determine the posterior probability distribution that the conclusion is correct.

In my opinion, the value in quantitative exercises with a BBN tool is not in the calculation of specific probability estimates (since the priors and joint probability tables are difficult to elicit) but in exploring the contribution of different kinds of evidence, and beliefs about their influence (i.e., the probability tables), to assessment of system properties. In this example, BBN experiments reveal that changing the proportion of successful tests has a rather small impact on the posterior estimate of system correctness; its larger impact is on the posterior estimate of oracle correctness.

It is straightforward to include confidence claims and assumptions in BBNs. For example, we might have some means A to assess the "quality" of the formal specification Z; we would add A to the BBN of Figure 5.3, with an arc from Z to A (these items are drawn in blue) to indicate that A depends on Z, and a probability distribution for that dependency (e.g., we might suppose the probability that the assessment is satisfactory is 99% if Z is correct, but only 20% if it is incorrect). An arc from A to O indicates that our estimate of oracle correctness is influenced by the quality estimate, and the probability table for O will be changed to reflect this. Another arc, from A to C, indicates that A functions as a confidence claim that influences evaluation of the overall BBN: that is, in addition to successful outcomes for verification and testing, the specification must also pass the quality test. Experiments with a BBN tool will show that this particular A provides value only if it has a significant impact on correctness of the test oracle.[10]

5.3.2 Propagation of Probabilistic Assessment

BBNs allow us to explore our beliefs about the extent to which multiple items of evidence support a claim; ultimately, they may allow us to attach a single posterior probability to the claim. We can imagine doing this for all the evidential steps in an argument and might then hope to propagate these through the reasoning steps to arrive at an overall estimate for the quality or strength of the argument.

[10] This observation is interesting because, as we saw in Section 3.1 (particularly around [15]), many DO-178C objectives can be interpreted as confidence items that assess the quality of artifacts employed in evidence; it could be worthwhile to undertake BBN analysis of the relevant parts of DO-178C to investigate the extent to which these confidence items really do add value.

Unfortunately, probabilistic and deductive reasoning do not combine well. We might hope that if evidence E supports a claim C and C deductively entails C', then surely E should also support C'. This expectation is dashed by the following counterexample. Let C be the hypothesis that a certain playing card (drawn at random from a shuffled deck) is the Ace of Hearts, let C' be the claim that the card is red, and let E be the evidence that the card is an Ace. Certainly E (Ace) supports C (Ace of Hearts), which entails C' (red), but E (Ace) cannot be considered to support C' (red).[11] It is not clear how to find a general resolution for the difficulty illustrated by this example but one observation is that the evidence "Ace" provides incomplete support for the claim "Ace of Hearts"—this is really a conjunction "Ace AND Heart" and we have evidence for Ace but not for Heart. In an assurance case, we should surely demand evidence that covers all aspects of our claim and should be actively seeking defeaters that satisfy our evidence but violate our claim. Hence, we will discuss methods for propagating probabilistic assessments of evidential strength through the reasoning steps of an argument, even though the theoretical basis for doing so may not be completely firm. Our main motivation for doing this is to seek insight on the consequences of evidential weakening performed in the service of graduated assurance.

There a couple of ways in which probabilistic assessments generated (using BBNs) for the evidential steps of an argument might be propagated through the reasoning steps of a sound argument to deliver a quantitative assessment of confidence in the top level claim. The first approach is based on *Probability Logic*, whose purpose is to assess the probability of a conclusion derived (deductively) from premises that are probabilistically true. There are several treatments, among which Adams [1] is recent and widely cited. The central result is that the uncertainty (i.e., 1 minus the probability) of the conclusion is no worse than the sum of the uncertainties of the premises. Thus, if we have 10 leaf-level subclaims, each evidentially supported with probability 0.99 (i.e., uncertainty 0.01), then the top-level claim will have uncertainty less than 0.1, or probability greater than 0.9.

Adams' result is independent of the logical form of the reasoning steps. Another approach observes that reasoning steps are Horn clauses and the top-level claim therefore follows from the conjunction of leaf-level subclaims. If these are independent (and if they are not, their BBNs would have been entangled and the argument should have been revised), then the probability of the top claim is the product of the probabilities of the leaf-level claims. Thus our 10 leaf claims[12] with probability 0.99 yield a top claim of probability $0.99^{10} = 0.9$. This is similar to the "flattening" interpretation that we earlier decried (recall Figure 5.2); however, there

[11] This counterexample comes from a talk by Brandon Fitelson, then of UC Berkeley; his website http://fitelson.org/ contains much material on these topics.

[12] Observe that if we have 100 leaf-level subclaims, then Adams' value decays to 0 while the product yields 0.37.

the probabilistic analysis was proposed as a way to assess the overall quality of an argument, whereas here it is being used to measure the *strength* of an argument already assessed to be sound.

Both these treatments and their estimates are too crude and pessimistic to truly assess confidence in the top claim, but they reinforce the (admittedly obvious) point that this will be dominated by the "weakest link" in the evidential support for leaf-level subclaims. For example, if one of the 10 evidential steps considered earlier has its probability weakened from 0.99 to 0.5 then the overall assessment collapses to 0.41 under Adams' method and 0.46 by the product method. On the other hand, if all 10 are weakened to 0.95 then the overall assessment becomes 0.5 under Adams' method and 0.6 by the product method.

Thus a recommendation could be made that graduated assurance should be achieved by lowering the threshold by a little across all leaf-level subclaims rather than by lowering it significantly for a few. A counterargument is that this ignores the logical contribution of each subclaim: surely some are more important than others. It is a topic for future research to explore how this might be accommodated.

5.4 Confidence in Assurance Cases and Certification

We have seen how it may be possible to evaluate the soundness of an assurance case, and how it may even be possible to assign a numerical "strength" to the evaluation: that is, a subjective probability that expresses our confidence in the soundness of the case.

But what we are really interested in is not the assurance case but the system it is about, and so we need to ask how confidence in the assurance case translates to confidence in the property assured for the system. And, more particularly, how a probabilistic measure of confidence in the assurance case translates to a probabilistic measure—presumably probability of failure of the assured property—about the system.

Our analysis of this topic is slightly different for software assurance cases (where the top claim is about correctness) and system assurance cases (where the top claim is about a property of the system, and may itself be expressed probabilistically). Our treatment of system assurance cases builds on that for software cases and so we start with the latter.

5.4.1 Probabilistic Assessment for Software

The top-level claim for a software assurance case is that the executable object code (EOC) is *correct* with respect to the system requirements. Confidence in a software assurance case can be expressed as a subjective probability that the software concerned really is correct:

$$\text{Confidence in software assurance case} = P(\text{software correct}).$$

By the formula for total probability

$$\begin{aligned}
&P(\text{software fails [on a randomly selected demand]}) \\
&\quad = P(\text{software fails} \,|\, \text{software correct}) \times P(\text{software correct}) \\
&\qquad + P(\text{software fails} \,|\, \text{software incorrect}) \times P(\text{software incorrect}).
\end{aligned} \tag{5.8}$$

Now, failure here means "violates system requirements": it is a separate task (namely, that of the system assurance case) to provide assurance that the system requirements ensure the top-level claim—e.g., safety—for the system. Hence, the first term in the sum on the right hand side of (5.8) is zero, because the software does not fail, in this sense, if it is correct.

To make the notation consistent with previous treatments [110], we will use *nonfaulty* as a synonym for correct (and *faulty* for incorrect). We define p_{nf} as the probability the software is nonfaulty, so that $P(\text{software incorrect}) = 1 - p_{nf}$, and $p_{F|f}$ as the probability that it Fails, if faulty, then (5.8) becomes

$$P(\text{software fails [on a randomly selected demand]}) = p_{F|f} \times (1 - p_{nf}).$$

More importantly, $p_{srv}(n)$, the probability of surviving n independent demands without failure is given by

$$p_{srv}(n) = p_{nf} + (1 - p_{nf}) \times (1 - p_{F|f})^n. \tag{5.9}$$

A suitably large n can represent "the entire lifetime of all aircraft of one type." For example, if we regard a complete flight as a demand, then n might be in the range 10^8–10^9 (aircraft of the Airbus A320 series have already performed over 62 million—i.e., approaching 10^8—flights). The notable feature of (5.9) is that the first term establishes a lower bound for $p_{srv}(n)$ that is independent of n. Thus, if software assurance gives us the confidence to assess, say, $p_{nf} > 0.99$, (or whatever threshold "not expected to occur" means) then it looks like we have sufficient evidence to certify the aircraft as safe (with respect to software aspects).

But certifiers, and the public, will also want guarantees in case the software does have faults. Thus, we need confidence that the second term in (5.9), which decays exponentially, will be well above zero. This confidence could come from prior failure-free operation (e.g., flight tests). Calculating the overall $p_{srv}(n)$ can then be posed as a problem in Bayesian inference: we have assessed a value for p_{nf}, have observed some number r of failure-free demands, and want to predict the probability of seeing $n - r$ future failure-free demands. To do this, we need a prior distribution for $p_{F|f}$, which may be difficult to obtain, and difficult to justify for certification. However, there is a distribution that delivers *provably worst-case* predictions (specifically, one in which $p_{F|f}$ is concentrated in a probability mass at some $q_n \in (0, 1]$) [117] so we can make predictions that are *guaranteed* to be conservative given only p_{nf}, r, and n. For values of p_{nf} above 0.9, we find that $p_{srv}(n)$ is well above the floor given by p_{nf}, provided $r > \frac{n}{10}$.

We need an estimate of $p_{srv}(n)$ for n as large as 10^9, but flight tests prior to certification might provide only $r = 10^3$, so it looks as if these are insufficient for certification by the criterion above. However, it can be argued that when an aircraft type is certified, we do not require (and in fact cannot feasibly obtain) sufficient evidence to predict failure-free operation over the entire lifetime of the type; instead, we initially require sufficient confidence for only, say, the first six months of operation and the small number of aircraft that will be deployed in that period. This will be a much smaller value of n, and our p_{nf} (from assurance) and our r (from flight tests) will be sufficient for confidence in its failure-free operation. Then we will need confidence in the next six months of operation, with a larger fleet, (i.e., a larger n) but now have the experience of the prior six months failure-free operation (i.e., a larger r) and in this way we can "bootstrap" our way forward.

We contend that the account of the previous paragraph provides the first scientific explanation for the way software certification in some industries (particularly commercial aviation) actually works: that is, it provides a model for beliefs and their quantification that explains and is consistent with current practice.

5.4.2 Probabilistic Assessment for Systems

System assurance cases differ from software cases in that the top claim is not correctness but the system-level property of interest, such as safety. Furthermore, the claim may be stated probabilistically, as in "no catastrophic failure condition is expected in 10^9 flights." We now have to ask how confidence in the assurance case, expressed probabilistically, combines with the top claim, also expressed probabilistically, to yield some overall expression of confidence in the system.

Now, although the top claim of the assurance *case* may be stated probabilistically, I do not think this is likely to be so for the *argument* of the case. For example, the top claim of the case may be "no catastrophic failure condition is expected in 10^9 flights," but the top claim of the argument will be the categorical "there will be no catastrophic failure condition." However, the argument will employ assumptions that are contingent on uncertainties in the world; typically, these will relate to random failures and exhaustion of redundancy, such as "the number of failed accelerometers is at most 3." The system assurance case is based on the truth of these assumptions and its evaluation likewise assumes their truth (possibly following challenge, revision, and further review). The top claim of the assurance case then follows by allocating probabilities to these assumptions and propagating them appropriately.

Let E be the categorical top claim of the system assurance argument (e.g., "no catastrophic failure condition"), then the top claim of the assurance case will be some probabilistic statement about this event, such as its *pfd* or, more likely, its probability of holding over a long run: we can let $p_{srv}(E, n)$ denote the probability of E holding over n demands (e.g., flights). The system assurance case constructs an estimate for this quantity, assuming the assurance case argument is sound: $p_{srv}(E, n \,|\, \text{argument is sound})$ which we can denote by $p_{srv|nf}$. By the law of total probability, we will have

$$\begin{aligned} & p_{srv}(E, n) \\ & \quad = \quad p_{srv}(E, n \,|\, \text{argument is sound}) \times P(\text{argument is sound}) \\ & \qquad\quad + p_{srv}(E, n \,|\, \text{argument is unsound}) \times P(\text{argument is unsound}). \end{aligned} \tag{5.10}$$

As in the case of software assurance arguments, confidence in a system argument, expressed here as $P(\text{argument is sound})$, is a subjective probability that the system is correct, or nonfaulty, for E: that is, the probability that E will never fail, no matter how much exposure it has. To reuse notation from the software case, we can denote this by p_{nf}. Likewise by analogy to the software case, we can use $p_{F|f}$ to denote the probability that the system fails for E (on a random demand), if it is faulty. Equation (5.10) then becomes

$$\begin{aligned} & p_{srv}(E, n) \\ & \quad = \quad p_{srv|nf} \times p_{nf} \\ & \qquad\quad + (1 - p_{F|f})^n \times (1 - p_{nf}). \end{aligned} \tag{5.11}$$

This is essentially the same as the software case (5.9), except the first term in the sum is $p_{srv|nf} \times p_{nf}$ rather than simply p_{nf}.

Note that $p_{srv|nf}$ is a probabilistic assessment that assumes the case is sound but factors in probabilistic assessments for random events in the world (e.g., redundancy exhaustion), while p_{nf} is a probabilistic assessment of the soundness of the case itself.

As with the software case, our evaluation yields an estimate for the first term in this sum, while flight tests and incremental experience can be used to bootstrap confidence in the second.

Chapter 6

Conclusion

Assurance Cases are a significant contribution to system and software assurance and certification. The principal advance offered by assurance cases compared to other forms of assurance is provision of an explicit argument connecting evidence to claims. The argument provides a context in which to justify and assess the quality and relevance of the evidence submitted, and the soundness of the reasoning that relates this to the hierarchy of subclaims that leads to the top-level claim. In contrast, traditional guidelines such as DO-178C specify the evidence to be submitted, but do not document the rationale for its selection.

For the first time, it may now be feasible to assess whether the basis for certification is intellectually sound. That is, assurance cases provide a framework in which assurance and certification can be set on a rigorous footing—although there are several issues and details that must be resolved before that promise can be realized. Principal among these is the need to provide a solid interpretation for inductive reasoning steps and associated confidence claims, or to demonstrate that realistic cases can be constructed in which the reasoning (i.e, interior) steps are strictly deductive; related topics include improvement in the languages and notations used to present assurance cases and the tool support for these. Examples of these topics include explication of the interpretation for backing nodes in CAE and for contexts in GSN; annotations to indicate whether argument steps are to be interpreted as deductive, inductive, or evidential; integration with other tools and documents under configuration management; tool-supported assessment of simple properties such as completeness, circularity and local notions of logical validity; and tool support (in the sense of recording and managing, not evaluation) for challenging and probing cases using postulated defeaters or by other means.

I advocate that assurance cases are structured in "simple form" where each argument step is supported either by subclaims (i.e., an interior or reasoning step) or by evidence (i.e., a leaf or evidential step), but not by a mixture (although we can allow evidential steps in which subclaims are used as assumptions). The reason is that the two kinds of argument step should be interpreted differently. Evidential steps connect the real world of measurement and observation to the conceptual world of claims: they are the bridge between epistemology and logic, and are quintessentially inductive (i.e., they can strongly support but seldom guarantee their claim). Multiple items of evidence are interpreted *cumulatively*: informally they are "weighed in the balance" and if their combined weight crosses some threshold we treat their claim as a "settled fact"; more formally, this accumulation can be understood within the probabilistic framework of Bayesian Epistemology and evaluated and explored using BBNs, although the feasibility and benefit of doing this awaits practical demonstration.

Interior argument steps are about reasoning: they should be interpreted in logic, not epistemology. Multiple subclaims supporting a claim are interpreted as a *conjunction*. My own preference is for assurance cases in which the reasoning steps are deductive, meaning the conjunction of subclaims necessarily entails the supported claim. I know no good semantics for reasoning steps that are inductive, where the subclaims merely suggest the claim, and no good interpretation for confidence claims in reasoning steps. Defeasible logics and argumentation structures do provide semantics for closely related notions, but my opinion is that assurance cases are a sufficiently distinct context that their treatments are unsuitable.

Current practice accepts inductive reasoning steps and the use of confidence arguments, so recommending that these should be eschewed may regarded as impractical at best, and misguided at worst. Certainly, it seems challenging to structure large system assurance cases so that all interior steps are deductive, but we note that current cases often mix epistemic accumulation (which is necessarily inductive) with logical conjunction. Hence, a first measure would be to restructure such cases into simple form, which can be performed algorithmically—as exemplified in Figure 3.2 on Page 41. We suspect that many inductive elements (and associated confidence claims) will migrate to the evidential steps and the interior steps will be simplified. It will then be feasible to review the extent to which interior reasoning steps are, and should remain, inductive. Especially for software, where the top-level claim is correctness rather than safety, we believe that purely deductive reasoning steps is a feasible and worthwhile aim. Others may believe that forcing all interior steps to be deductive will distort the argument and render it less rather than more understandable and persuasive. The only way to evaluate the merits of our recommendation and objections to it is by experiment—perhaps by reworking some existing cases.

Confirmation bias is an ineradicable human propensity, so evaluation of an assurance case requires effective external review and vigorous challenge. Tool support for assurance cases should support challenge and active forms of review, not mere browsing. Reviewers should be able to readily discover the evidential support for any claim, and the consequences of denying or perturbing a claim, item of evidence, or argument step. Some of the techniques from defeasible reasoning and dialectics could be useful here during active review, but I am skeptical that such techniques should be used to evaluate the final case: defeasible reasoning and dialectics are motivated by reasoning under uncertainty or in contested domains, or to explore human reasoning; assurance cases are a different type of application, where the goal is to develop an unimpeachable argument based on agreed evidence.

Although guidelines and standards might not have such a sound intellectual foundation as assurance cases, they do seem to work, and one of the reasons may be the intense challenge and scrutiny these undergo in committee. It is reasonable to be concerned that a bespoke assurance case may not receive such effective scrutiny.

It is plausible that assurance cases could largely be assembled in a modular fashion from pre-developed and "validated" component subcases. The blocks and patterns developed for CAE and GSN are significant steps in this direction. The danger, of course, is that safety and other critical properties are seldom compositional (i.e., the safety of parts does not imply safety of the whole) so this may merely replace concern about the inner details of subcases by concern for the relevance and appropriateness of a pre-developed subcase in its specific context. Nonetheless, this is a fertile area for investigation. Assurance is currently a craft, where apprentices must learn by experience at the knee of a master; predefined subcases would help codify the "lore" and enable the transition from craft to engineering. What seems desirable is the rigor of CAE blocks combined with the larger scope of GSN patterns.

I recommend that guidelines and standards should be restructured as assurance cases: at first retrospectively, as with Michael Holloway's Explicate'78 project [61–63], and then prospectively, as they are revised. This can do no harm—indeed, might improve them—but would make it much easier to adapt them to new contexts (e.g., new kinds of systems, such as low-altitude UAS, new combinations of systems, such as NextGen, and new methods of assurance, such as sound or unsound static analysis), and would also provide a source of "validated subcases" for the construction of bespoke cases.

A more aggressive proposal (which I first heard from John Knight [77]) calls for wholesale replacement of existing standards—for example, DO-178C (software) and all its supplements, DO-254 (complex hardware), DO-197 (Integrated Modular Avionics), DO-278A (Air Traffic Management), ARP 4754A (aircraft systems)—by a single generic assurance case for "design assurance" plus guidelines on "best practice" for its instantiation and application to different technical areas. This would couple well with another aggressive proposal (which I first heard from Alan Wassyng) that fully bespoke assurance cases should be discouraged and instantiation of standardized templates encouraged. The reason for this is that it is difficult to avoid confirmation bias and hard to obtain really effective review for fully bespoke cases, whereas standardized templates could draw on the same dialectical committee processes that seem to be quite effective for conventional standards and guidelines. I am sympathetic to these proposals.

I regret the graphical focus of current assurance case notations such as CAE and GSN: I accept that diagrams can be effective means of communication (at least for small problems, or for fragments of larger ones) but would prefer to see these as renderings of a textual base language having a well-defined semantics that pays full attention to issues such as scoping of names, and the logical context in which claims are interpreted. Such a language could have graphical renderings that (exactly) resemble CAE or GSN, but would provide a much stronger basis for automated support and analysis.

I am intrigued by futuristic ideas expressed in the DEOS [121] and AMADEOS [5] projects and by the SM@RT (Safety Models at RunTime) community [123], which anticipate that safety-critical systems might assemble dynamically (as, for example, when an agricultural implement attaches to a tractor, or a pulse oximeter is connected to the same patient as an analgesia-dispensing infusion pump). Conceptually, the component systems exchange their assurance cases and a case for the composed system is calculated at runtime [107, 108], possibly with automated synthesis of protective wrappers—which is feasible (if, indeed, it is feasible) only if the constituent cases are represented in a form that supports automated interpretation and analysis. Similar ideas can be used to "update" an assurance case when the system is modified or its environment changes [29].

Finally, I would observe that construction and evaluation of a system assurance case is one of the most challenging and intellectually stimulating of human endeavors; it draws on topics that have occupied mankind since the dawn of philosophical inquiry—argumentation, logic, and epistemology—but aims to reintegrate and apply these at industrial scale, and to partially automate their support and evaluation. This is a rich field for theoretical and practical research and socially productive application.

Bibliography

[1] Ernest Wilcox Adams. *A Primer of Probability Logic*. Center for the Study of Language and Information (CSLI), Stanford University, 1998. 105

[2] *ASCAD: Adelard Safety Case Development Manual*. Adelard LLP, London, UK, 1998. Available from `http://www.adelard.com/resources/ascad/`. 21, 55

[3] Leonard Adelman, Paul E. Lehner, Brant A. Cheikes, and Mark F Taylor. An empirical evaluation of structured argumentation using the Toulmin argument formalism. *IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans*, 37(3):340–347, May 2007. 85, 86

[4] M. Anthony Aiello, Ashlie B. Hocking, John Knight, and Jonathan Rowanhill. SCT: A safety case toolkit. In *ASSURE: Second International Workshop on Assurance Cases for Software-Intensive Systems* [67], pages 216–219. 70

[5] *Basic SoS Concepts, Glossary and Preliminary Conceptual Model*. AMADEOS Project, June 2014. `http://amadeos-project.eu/documents/public-deliverables/`. 114

[6] B. Scott Andersen and George Romanski. Verification of safety-critical software. *Communications of the ACM*, 54(10):52–57, 2011. 26

[7] Anaheed Ayoub, Jian Chang, Oleg Sokolsky, and Insup Lee. Assessing the overall sufficiency of safety arguments. In Chris Dale and Tom Anderson, editors, *Assuring the Safety of Systems: Proceedings of the 21st Safety-Critical Systems Symposium*, pages 127–144, Bristol, UK, February 2013. 93

[8] Roger E. Bilstein. *Orders of Magnitude: A History of the NACA and NASA, 1915-1990*. The NASA History Series, Washington DC, November 1989. NASA/SP-4406. 15

[9] Robin Bloomfield and Peter Bishop. Safety and assurance cases: Past, present and possible future—an Adelard perspective. In Chris Dale and Tom Anderson, editors, *Advances in System Safety: Proceedings of the Nineteenth*

Safety-Critical Systems Symposium, pages 51–67, Bristol, UK, February 2010. 21, 55

[10] Robin Bloomfield, Sofia Guerra, and Kateryn Netkachova. *Guidance on CAE: Concepts, Blocks, Templates.* Adelard LLP, London, UK, April 2014. Draft. 59, 61

[11] 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. 102

[12] Robin Bloomfield and Kateryn Netkachova. Building blocks for assurance cases. In *ASSURE: Second International Workshop on Assurance Cases for Software-Intensive Systems* [67], pages 186–191. 56, 57, 60

[13] Luc Bovens and Stephan Hartmann. *Bayesian Epistemology.* Oxford University Press, 2003. 92

[14] BPMN. *OMG Buiness Process Model and Notation BPMN home page.* http://www.omg.org/spec/BPMN. 74

[15] V. Cassano and T. S. E. Maibaum. The definition and assessment of a safety argument. In *ASSURE: Second International Workshop on Assurance Cases for Software-Intensive Systems* [67], pages 180–185. 86

[16] Carlos Chesñevar et al. Towards an argument interchange format. *The Knowledge Engineering Review*, 21(4):293–316, 2006. 72, 87

[17] Carlos Iván Chesñevar, Ana Gabriela Maguitman, and Ronald Prescott Loui. Logical models of argument. *ACM Computing Surveys*, 32(4):337–383, 2000. 83

[18] L. Jonathan Cohen. *The Probable and The Provable.* Clarendon Press, Oxford, UK, 1977. 99

[19] L. Jonathan Cohen. On the psychology of prediction: Whose is the fallacy? *Cognition*, 7:385–407, 1979. 99

[20] L. Jonathan Cohen. Whose is the fallacy? a rejoinder to Daniel Kahneman and Amos Tversky. *Cognition*, 8:89–92, 1980. 99

[21] Simon Cruanes, Grégoire Hamon, Sam Owre, and Natarajan Shankar. Tool integration with the Evidential Tool Bus. In Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni, editors, *Verification, Model Checking, and Abstract*

Interpretation, 14th International Conference, VMCAI 2013, Volume 7737 of Springer-Verlag *Lecture Notes in Computer Science*, pages 275–294, Rome, Italy, January 2013. 74

[22] Simon Cruanes, Stijn Heymans, Ian Mason, Sam Owre, and Natarajan Shankar. The semantics of Datalog for the Evidential Tool Bus. In S. Iida, J. Meseguer, and K. Ogata, editors, *Specification, Algebra, and Software, A Festschrift Symposium in Honor of Kokichi Futatsugi*, Volume 8373 of Springer-Verlag *Lecture Notes in Computer Science*, pages 256–275, Kanazawa, Japan, April 2014. 74

[23] The Hon. Lord William Douglas Cullen. The public inquiry into the Piper Alpha disaster. Report, The Stationery Office, London, UK, November 1990. Two volumes. 19

[24] The Hon. Lord William Douglas Cullen. The development of safety legislation. Royal Academy of Engineering and Royal Society of Edinburgh Lecture, 1996. Available at http://www.scribd.com/doc/57924918/THE-DEVELOPMENT-OF-SAFETY-LEGISLATION. 14

[25] D-Case. *D-Case Editor home page*. http://www.dcase.jp/editor_en.html. 71

[26] Ewen Denney, Dwight Naylor, and Ganesh Pai. Querying safety cases. In SafeComp *2014: Proceedings of the 33rd International Conference on Computer Safety, Reliability, and Security*, Volume 8666 of Springer-Verlag *Lecture Notes in Computer Science*, pages 294–309, Florence, Italy, September 2014. 70

[27] Ewen Denney and Ganesh Pai. Automating the assembly of aviation safety cases. *IEEE Transactions on Reliability*, 63(4):830–849, December 2014. 70

[28] Ewen Denney, Ganesh Pai, and Ibrahim Habli. Towards measurement of confidence in safety cases. In *Fifth International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 380–383, IEEE Computer Society, Banff, Canada, September 2011. 93

[29] Ewen Denney, Ganesh Pai, and Ibrahim Habli. Dynamic safety cases for through-life safety assurance. In *Proceedings of the 37th International Conference on Software Engineering (ICSE), New Ideas and Emerging Results Track (NIER)*, Florence, Italy, May 2015. 114

[30] Ewen Denney, Ganesh Pai, and Joe Pohl. AdvoCATE: An assurance case automation toolset. In *Proceedings of the Workshop on Next Generation of*

System Assurance Approaches for Safety Critical Systems (SASSUR), Magdeburg, Germany, September 2012. 70

[31] Ewen Denney, Ganesh Pai, and Iain Whiteside. Hierarchical safety cases. In *NASA Formal Methods*, Volume 7871 of Springer-Verlag *Lecture Notes in Computer Science*, pages 478–483, Mountain View, CA, May 2013. 70

[32] George Despotou. *GSN Reference Card*. University of York, UK, v1.2 edition, 2010. Available at http://bit.ly/cEWA5B. 62

[33] Homayoon Dezfuli et al. *NASA System Safety Handbook Volume 1, System Safety Framework and Concepts for Implementation*. NASA Headquarters, Washington DC, November 2011. NASA/SP-2010-580. 23

[34] John Earman. *Bayes or Bust? A Critical Examination of Bayesian Confirmation Theory*. MIT Press, 1992. 92

[35] John Earman, John Roberts, and Sheldon Smith. *Ceteris Paribus* lost. *Erkenntnis*, 57(3):281–301, 2002. 91

[36] Luke Emmet. *Quantitative Confidence Propagation Across ASCE Networks?* Adelard LPP, London, UK, December 2013. Slide presentation at http://www.adelard.com/asce/user-group/4-Dec-2013/ASCE_confidence_propagation%20-%20Luke%20Emmet.pdf. 62

[37] Luke Emmet and George Cleland. Graphical notations, narratives and persuasion: A pliant systems approach to hypertext tool design. In *Thirteenth ACM Conference on Hypertext and Hypermedia*, pages 55–64, ACM, College Park, MD, June 2002. 62

[38] *Certification Specifications and Acceptable Means of Compliance for Large Aeroplanes, CS-25 and AMC-25*. The European Aviation Safety Agency (EASA), July 2011. Amendment 11; available at http://www.easa.eu.int/agency-measures/certification-specifications.php#CS-25. 16

[39] *Conducting Software Reviews Prior to Certification*. FAA Aircraft Certification Service, rev 1 edition, January 2004. Software Review Job Aid. 26

[40] Saeed Fararooy. Managing a system safety case in an integrated environment. Available at http://www.iscade.co.uk/downloads/rcm2sf_rhas_paper.pdf. 70

[41] *Special Conditions: Boeing Model 787-8 Airplane; Systems and Data Networks Security-Protection of Airplane Systems and Data Networks from Unauthorized External Access*. Federal Aviation Administration, December 28, 2007. Listed in the Federal Register on date shown. 16

[42] Branden Fitelson. *Studies in Bayesian Confirmation Theory.* PhD thesis, Department of Philosophy, University of Wisconsin, Madison, May 2001. Available at http://fitelson.org/thesis.pdf. 92

[43] John B. Goodenough, Charles B. Weinstock, and Ari Z. Klein. Eliminative argumentation: A basis for arguing confidence in system properties. Technical Report CMU/SEI-2014-TR-013, Carnegie Mellon Software Engineering Institute, Pittsburgh, PA, 2014. 98, 99

[44] Thomas F. Gordon, Henry Prakken, and Douglas Walton. The Carneades model of argument and burden of proof. *Artificial Intelligence*, 171(10):875–896, 2007. 87

[45] Patrick J. Graydon, John C. Knight, and Elisabeth A. Strunk. Assurance-based development of critical systems. In *The International Conference on Dependable Systems and Networks*, pages 347–357, IEEE Computer Society, Edinburgh, Scotland, June 2007. 67, 74

[46] Patrick John Graydon. Towards a clearer understanding of context and its role in assurance argument confidence. In SafeComp *2014: Proceedings of the 33rd International Conference on Computer Safety, Reliability, and Security*, Volume 8666 of Springer-Verlag *Lecture Notes in Computer Science*, pages 139–154, Florence, Italy, September 2014. 65

[47] William S. Greenwell, John C. Knight, C. Michael Holloway, and Jacob J. Pease. A taxonomy of fallacies in system safety arguments. In *Proceedings of the 24th International System Safety Conference*, Albuquerque, NM, 2006. 81

[48] *GSN Community Standard Version 1.* GSN Working Group, York, UK, 2011. Available at http://www.goalstructuringnotation.info/. 47, 63, 64, 65, 69

[49] Charles Haddon-Cave. The Nimrod Review: An independent review into the broader issues surrounding the loss of the RAF Nimrod MR2 Aircraft XV230 in Afghanistan in 2006. Report, The Stationery Office, London, UK, October 2009. Available at http://www.official-documents.gov.uk/document/hc0809/hc10/1025/1025.pdf. 95, 99

[50] Charles B. Haley, Robin Laney, Jonathan D. Moffett, and Bashar Nuseibeh. Security requirements engineering: A framework for representation and analysis. *IEEE Transactions on Software Engineering*, 34(1):133–153, 2008. 97, 98

[51] David Hamilton, Rick Covington, and John Kelly. Experiences in applying formal methods to the analysis of software and system requirements. In *WIFT*

'95: Workshop on Industrial-Strength Formal Specification Techniques, pages 30–43, IEEE Computer Society, Boca Raton, FL, 1995. 33

[52] Code of Hammurabi, c. 1772 BC. English translation by L.W. King available at http://eawc.evansville.edu/anthology/hammurabi.htm. 13

[53] Richard Hawkins, Kester Clegg, Rob Alexander, and Tim Kelly. Using a software safety argument pattern catalogue: Two case studies. In SafeComp *2011: Proceedings of the 30th International Conference on Computer Safety, Reliability, and Security*, volume 6894 of *Lecture Notes in Computer Science*, pages 185–198. Springer-Verlag, Naples, Italy, September 2011. 66, 67

[54] Richard Hawkins and Tim Kelly. A systematic approach for developing software safety arguments. *Journal of System Safety*, 46(4):25–33, 2010. 66

[55] Richard Hawkins, Tim Kelly, John Knight, and Patrick Graydon. A new approach to creating clear safety arguments. In Chris Dale and Tom Anderson, editors, *Advances in System Safety: Proceedings of the Nineteenth Safety-Critical Systems Symposium*, pages 3–23, Southampton, UK, February 2011. 68, 69, 89, 90

[56] James Hawthorne. Bayesian induction IS eliminative induction. *Philosophical Topics*, 21(1):99–138, 1993. 98

[57] *Using Safety Cases in Industry and Healthcare*. The Health Foundation, London UK, December 2012. 22

[58] Carl G. Hempel. Provisoes: A problem concerning the inferential function of scientific theories. *Erkenntnis*, 28:147–164, 1988. Also in conference proceedings “The Limits of Deductivism,” edited by Adolf Grünbaum and W. Salmon, University of California Press, 1988. 91

[59] E. G. Hiller. On the working of the boiler explosions acts 1882 and 1890. *Transactions of the Institution of Mechanical Engineers*, xvii:19–53, 1898. 14

[60] C. Michael Holloway. Safety case notations: Alternatives for the non-graphically inclined? In *3rd IET International Conference on System Safety*, The Institutions of Engineering and Technology, Birmingham, UK, October 2008. 47

[61] C. Michael Holloway. Towards understanding the DO-178C/ED-12C assurance case. In *7th IET International Conference on System Safety*, The Institution of Engineering and Technology, Edinburgh, UK, October 2012. 113

[62] C. Michael Holloway. Making the implicit explicit: Towards an assurance case for DO-178C. In *Proceedings of the 31st International System Safety Conference*, Boston, MA, August 2013. 101, 113

[63] C. Michael Holloway. Explicate '78: Discovering the implicit assurance case in DO-178C. In Mike Parsons and Tom Anderson, editors, *Engineering Systems for Safety. Proceedings of the 23rd Safety-critical Systems Symposium*, pages 205–225, Bristol, UK, February 2015. 3, 10, 101, 113

[64] Alfred Horn. On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic*, 16(1):14–21, 1951. 29

[65] HUGIN Expert. *Hugin home page*, Retrieved 2015. `http://www.hugin.com/`. 104

[66] Patrick J. Hurley. *A Concise Introduction to Logic*. Cengage Learning, 12th edition, 2015. 22

[67] *ASSURE: Second International Workshop on Assurance Cases for Software-Intensive Systems*, Naples, Italy, November 2014. IEEE International Symposium on Software Reliability Engineering Workshops. 115, 116

[68] J. R. Inge. The safety case: Its development and use in the United Kingdom. In *Equipment Safety Assurance Symposium*, 2007. 20

[69] *IEC 61508—Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems*. International Electrotechnical Commission, Geneva, Switzerland, March 2004. Seven volumes; see `http://www.iec.ch/zone/fsafety/fsafety_entry.htm`. 47

[70] Richard Jeffrey. *Subjective Probability: The Real Thing*. Cambridge University Press, 2004. 102

[71] Leslie A. (Schad) Johnson. DO-178B, "Software Considerations in Airborne Systems and Equipment Certification". *Crosstalk*, October 1998. See `http://www.crosstalkonline.org/back-issues/` and `http://www.dcs.gla.ac.uk/~johnson/teaching/safety/reports/schad.html`. 17, 18

[72] James M. Joyce. On the plurality of probabilist measures of evidential relevance. In *Bayesian Epistemology Workshop of the 26th International Wittgenstein Symposium*, Kirchberg, Austria, August 2003. Available at `http://www.uni-konstanz.de/ppm/kirchberg/Joyce_1.pdf`. 92

[73] Jonathan S. Kahan. Premarket approval versus premarket notification: Different routes to the same market. *Food, Drug, Cosmetic Law Journal*, 39:510–525, 1984. 60

[74] Daniel Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, 2011. 80

[75] Daniel Kahneman and Amos Tversky. On the interpretation of intuitive probability: A reply to Johnathan Cohen. *Cognition*, 7:409–411, 1979. 99

[76] Tim Kelly. *Arguing Safety—A Systematic Approach to Safety Case Management*. DPhil thesis, Department of Computer Science, University of York, UK, 1998. 21, 62, 66, 67

[77] John C. Knight. *Advances in Software Technology Since 1992 and a Modest Proposal for their Incorporation Into Certification*. University of Virgina, 2008. Presented to annual FAA Software Assurance workshop; available at http://www.cs.virginia.edu/7Ejck/publications/FAA.SW.AEH.2008.PDF. 113

[78] Kimio Kuramtisu. D-Script: Dependable scripting with DEOS process. In *3rd International Workshop on Open Systems Dependability (WOSD)*, pages 326–330, Pasadena, CA, November 2013. Workshop held in association with ISSRE'13. 71

[79] Nancy Leveson. The use of safety cases in certification and regulation. *Journal of System Safety*, 47(6):1–5, 2011. 95

[80] 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. 102

[81] Yukata Matsuno. A design and implementation of an assurance case language. In *The International Conference on Dependable Systems and Networks*, pages 630–641, IEEE Computer Society, Atlanta, GA, June 2014. 71

[82] Yukata Matsuno. Design and implementation of GSN patterns: A step toward assurance case language. *Information Processing Society of Japan, Transactions on Programming*, 7(2):1–10, 2014. 67, 71

[83] Catherine Menon, Richard Hawkins, and John McDermid. Interim standard of best practice on software in the context of DS 00-56 Issue 4. Document SSEI-BP-000001, Software Systems Engineering Initiative, University of York, UK, August 2009. 66

[84] Hugo Mercier and Dan Sperber. Why do humans reason? Arguments for an argumentative theory. *Behavioural and Brain Sciences*, 34(2):57–111, 2011. See also the commentary on page 74 by Roy F. Baumeister, E. J. Masicampo,

and C. Nathan DeWall: "Arguing, Reasoning, and the Interpersonal (Cultural) Functions of Human Consciousness". 95

[85] Susan Newman and Catherine Marshall. Pushing Toulmin too far: Learning from an argument representation scheme. Technical Report SSL-92-45, Xerox PARC, Palo Alto, CA, 1992. 86

[86] Sven Nordhoff. *DO-178C/ED-12C*. SQS Software Quality Systems, Cologne, Germany, Undated. White Paper available at http://www.sqs.com/us/_download/DO-178C_ED-12C.pdf. 26

[87] *Infusion Pumps Total Product Life Cycle Guidance for Industry and FDA Staff*. Office of Device Evaluation, Center for Devices and Radiological Health, Food and Drug Administration, December 2014. Available at http://www.fda.gov/downloads/MedicalDevices/DeviceRegulationandGuidance/GuidanceDocuments/UCM209337.pdf. 22

[88] Sascha Ossowski, editor. *Agreement Technologies*. Law, Governance and Technology Series, vol. 8. Springer, 2013. 83, 87

[89] D. L. Parnas and P. C. Clements. A rational design process: How and why to fake it. *IEEE Transactions on Software Engineering*, SE-12(2):251–257, February 1986. Correction: Aug 86, page 874. 26

[90] Jeanne Peijnenburg. A case of confusing probability and confirmation. *Synthese*, 184(1):101–107, January 2012. 92

[91] John L. Pollock. *Cognitive Carpentry: A Blueprint for How to Build a Person*. MIT Press, 1995. 98

[92] Karl Popper. *The Logic of Scientific Discovery*. Routledge, 2014. First published in German 1934, English 1959. 98

[93] John D. Ramage, John C. Bean, and June Johnson. *Writing Arguments: A Rhetoric With Readings*. Longman, 10th edition, 2015. 78

[94] RCM2 Limited. Specification for ISCaDE Pro. Available at http://www.iscade.co.uk/downloads/ISCaDEProWhitePaper.pdf. 70

[95] Felix Redmill. ALARP explored. Technical Report CS-TR-1197, Department of Computing Science, University of Newcastle upon Tyne, UK, March 2010. 60

[96] Chris Reed and Glenn Rowe. Araucaria: Software for argument analysis, diagramming and representation. *International Journal on Artificial Intelligence Tools*, 13(4):961–979, 2004. 87

[97] Chris Reed, Douglas Walton, and Fabrizio Macagno. Argument diagramming in logic, law and artificial intelligence. *The Knowledge Engineering Review*, 22(01):87–109, 2007. 21, 87

[98] *DO-178B: Software Considerations in Airborne Systems and Equipment Certification*. Requirements and Technical Concepts for Aviation (RTCA), Washington, DC, December 1992. This document is known as EUROCAE ED-12B in Europe. 18

[99] *DO-178C: Software Considerations in Airborne Systems and Equipment Certification*. Requirements and Technical Concepts for Aviation (RTCA), Washington, DC, December 2011. 2, 10, 19, 26, 27, 29

[100] *DO-330: Software Tool Qualification Considerations*. Requirements and Technical Concepts for Aviation (RTCA), Washington, DC, December 2011. 19

[101] *DO-331: Model-Based Development and Verification Supplement to DO-178C and DO-278A*. Requirements and Technical Concepts for Aviation (RTCA), Washington, DC, December 2011. 19

[102] *DO-332: Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A*. Requirements and Technical Concepts for Aviation (RTCA), Washington, DC, December 2011. 19

[103] *DO-333: Formal Methods Supplement to DO-178C and DO-278A*. Requirements and Technical Concepts for Aviation (RTCA), Washington, DC, December 2011. 19

[104] David Rinehart, John Knight, and Jonathan Rowanhill. Current practices in constructing and evaluating assurance cases with applications to aviation. NASA Contractor Report NASA/CR-2015-218678, NASA Langley Research Center, January 2015. Available at `http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20150002819.pdf`. i

[105] The Hon. Lord Alfred Robens et al. Safety and health at work: Report of the committee 1970-72. Report, The Stationery Office, London, UK, June 1972. 19

[106] J. Roxborough. Basic safety training for personnel. In *Safety and Health in the Oil and Gas Extractive Industries, Proceedings of an International Symposium*, pages 97–109, Commission of The European Communities, Luxembourg, April 1983. 14

[107] John Rushby. Just-in-time certification. In *12th IEEE International Conference on the Engineering of Complex Computer Systems (ICECCS)*, pages 15–24, IEEE Computer Society, Auckland, New Zealand, July 2007. 114

[108] John Rushby. Runtime certification. In Martin Leucker, editor, *Eighth Workshop on Runtime Verification: RV08*, Volume 5289 of Springer-Verlag *Lecture Notes in Computer Science*, pages 21–35, Budapest, Hungary, April 2008. 114

[109] John Rushby. New challenges in certification for aircraft software. In Sanjoy Baruah and Sebastian Fischmeister, editors, *Proceedings of the Ninth ACM International Conference On Embedded Software: EMSOFT*, pages 211–218, Association for Computing Machinery, Taipei, Taiwan, 2011. 26

[110] John Rushby, Bev Littlewood, and Lorenzo Strigini. Evaluating the assessment of software fault-freeness. In *Workshop on Assessing the Efficacy of Standards for Safety Critical Software (AESSCS 2014)*, Newcastle upon Tyne, UK, May 2014. Available at http://arxiv.org/find/all/1/all:+AESSCS/0/1/0/all/0/1. 107

[111] John Rushby, Xidong Xu, Murali Rangarajan, and Thomas L. Weaver. Understanding and evaluating assurance cases. NASA Contractor Report NASA/CR-2015-218802, NASA Langley Research Center, July 2015. i

[112] SACM. *OMG Structured Assurance Case Metamodel (SACM) home page.* http://www.omg.org/spec/SACM/. 72

[113] *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; revised as ARP 4754A, December 2010. 26

[114] *Aerospace Recommended Practice (ARP) 4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment.* Society of Automotive Engineers, December 1996. 26

[115] Mark Staples. Critical rationalism and engineering: Ontology. *Synthese*, 191(10):2255–2279, July 2014. 92

[116] Mark Staples. Critical rationalism and engineering: Methodology. *Synthese*, 192(1):337–362, January 2015. 92

[117] Lorenzo Strigini and Andrey Povyakalo. Software fault-freeness and reliability predictions. In SafeComp *2013: Proceedings of the 32nd International Conference on Computer Safety, Reliability, and Security*, Volume 8153 of Springer-Verlag *Lecture Notes in Computer Science*, pages 106–117, Toulouse, France, September 2013. 108

[118] Frederick Suppe. Hempel and the problem of provisos. In James H. Fetzer, editor, *Science, Explanation, and Rationality: Aspects of the Philosophy of Carl G. Hempel*, chapter 8, pages 186–213. Oxford University Press, 2000. 91

[119] Toshinori Takai and Hiroyuki Kido. A supplemental notation of GSN to deal with changes of assurance cases. In *4th International Workshop on Open Systems Dependability (WOSD)*, pages 461–466, IEEE International Symposium on Software Reliability Engineering Workshops, Naples, Italy, November 2014. 70, 71, 87

[120] Nguyen Thuy et al. Public case study: The stepwise shutdown system. Deliverable 5.4, European Commission HARMONICS Project, January 2015. Available at http://harmonics.vtt.fi/publications.htm. 57, 61

[121] Mario Tokoro. *Open Systems Dependability—Dependability Engineering for Ever-Changing Systems*. CRC Press, 2013. 71, 114

[122] Stephen Edelston Toulmin. *The Uses of Argument*. Cambridge University Press, 2003. Updated edition (the original is dated 1958). 21, 25, 55, 84

[123] Mario Trapp and Daniel Schneider. Safety assurance of open adaptive systems—a survey. In Nelly Bencomo, Robert France, Betty H.C. Cheng, and Uwe Assmann, editors, *Models@Run.Time: Foundations, Applications, and Roadmaps*, volume 8378 of *Lecture Notes in Computer Science*, pages 279–318. Springer-Verlag, 2014. 114

[124] *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. 22, 25, 78

[125] Sebastian Voss, Bernhard Schätz, Maged Khalil, and Carmen Carlan. Towards modular certification using integrated model-based safety cases, July 2013. Presented at VeriSure 2013, part of CAV; available at http://download.fortiss.org/public/projects/af3/research/2013/SafetyCasesinAF3.pdf. 71

[126] Carl G. Wagner. Review: The probably and the provable. by jonathan l. cohen. *Duke Law Journal*, 28(4):1071–1082, 1979. 99

[127] Alan Wassyng, Tom Maibaum, Mark Lawford, and Hans Bherer. Software certification: Is there a case against safety cases? In Radu Calinescu and Ethan Jackson, editors, *16th Monterey Workshop: Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*, Volume 6662 of Springer-Verlag *Lecture Notes in Computer Science*, pages 206–227, Redmond, WA, 2011. 95

[128] Rob A. Weaver. *The Safety of Software—Constructing and Assuring Arguments*. PhD thesis, Department of Computer Science, University of York, UK, 2003. 66

[129] Charles B. Weinstock, John B. Goodenough, and Ari Z. Klein. Measuring assurance case confidence using Baconian probabilities. In *1st International Workshop on Assurance Cases for Software-Intensive Systems (ASSURE)*, San Francisco, CA, May 2013. 98, 99

[130] S. P. Wilson, T. P. Kelly, and J. A. McDermid. Safety case development: Current practice, future prospects. In Roger Shaw, editor, *Safety and Reliability of Software Based Systems (Twelfth Annual CSR Workshop)*, pages 135–156, Bruges, Belgium, September 1995. 21, 62

[131] S. P. Wilson, J. A. McDermid, C. Pygott, and D. J. Tombs. Assessing complex computer based systems using the goal structuring notation. In *Seconf IEEE International Conference on the Engineering of Complex Computer Systems (ICECCS)*, pages 498–505, IEEE Computer Society, Montreal, Canada, October 1996. 21, 62

[132] David Wright and Kai-Yuan Cai. Representing uncertainty for safety critical systems. PRDCS Technical Report 135, City University, London, UK, May 1994. 93

[133] Fan Ye. *Justifying the Use of COTS Components within Safety Critical Applications*. PhD thesis, Department of Computer Science, University of York, UK, 2005. 66

[134] Fan Ye and George Cleland. *Weapons Operating Centre Approved Code of Practice for Electronic Safety Cases*. Adelard LLP, London, UK, March 2012. 61

[135] Tangming Yuan and Tim Kelly. Argument schemes in computer system safety engineering. *Informal Logic*, 31(2):89–109, 2011. 61

[136] Fuping Zeng, Manyan Lu, and Deming Zhong. Using D-S evidence theory to evaluation of confidence in safety case. *Journal of Theoretical and Applied Information Technology*, 47(1):184–189, January 2013. 93