Invited paper; appears in *Computer Safety, Reliability, and Security*: Proceedings of SafeComp 32, Toulouse, France, September 2013, Springer LNCS 8153, pp. 1–7.

Logic and Epistemology in Safety Cases

John Rushby

Computer Science Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025 USA

Abstract. A safety case must resolve concerns of two different kinds: how complete and accurate is our knowledge about aspects of the system (e.g., its requirements, environment, implementation, hazards) and how accurate is our reasoning about the design of the system, given our knowledge.
The first of these is a form of epistemology and requires human experience and insight, but the second can, in principle, be reduced to logic and then checked and automated using the technology of formal methods.
We propose that reducing epistemic doubt is the main challenge in safety cases, and discuss ways in which this might be achieved.

1 Introduction

Different industries take different approaches to software assurance and certification, but no matter what form a submission for certification actually takes, it can be understood and examined within the framework of a *safety case*. More specifically, we can suppose there are safety-relevant *claims* for the system concerned, some *evidence* about its assumptions, design, and construction, and an *argument*, based on the evidence, that the claims are satisfied. The standards and guidelines on which some certification processes are based often specify only the evidence to be produced; the claims and argument are implicit, but presumably informed the deliberations that produced the guidelines. There is current work to make explicit the safety cases implicit in some guidelines: for example, the FAA has sponsored work at NASA to do this for the airborne software guidelines DO-178C [1].

An assurance case serves two purposes: communication and reasoning. The first is about assisting human stakeholders to grasp the essence of the case, to explore its details, and to challenge it; the second is about being sure that the reasoning underlying the argument is sound and complete. The safety case for any interesting system will be large, and one wonders how reliable the processes of review and evaluation—and ultimately of certification—can be for such a large artifact, for either of its purposes. The Nimrod case demonstrates that these concerns are not idle [2]. In this paper, I propose that modern formal methods and the techniques of automated deduction allow the reasoning aspect of a safety case argument to be evaluated by systematic and largely automated methods.

That is to say, evaluation of a safety case argument can—and should—largely be reduced to calculation, in a modern application of Leibniz' vision:

> "If controversies were to arise, there would be no more need of disputation between two philosophers than between two accountants. For it would suffice to take their pencils in their hands, to sit down to their slates, and to say to each other... 'Let us calculate'."

This proposal is not intended to denigrate or displace human judgment and insight, but to liberate these from an essentially clerical task so that they may be focused on areas that really need them. In what follows, I hope to identify those areas where human judgment is best deployed, and how a formalized safety case can best be organized to facilitate this.

2 Logic Doubt

The argument of a safety case establishes claims, based on evidence, and this has obvious parallels with formal proof, which justifies a conclusion, based on premises. Modern methods of formal specification and verification provide notations adequate to the formalization of premises and conclusions concerning complex systems, and allow substantial automation to be applied to the construction of formal proofs. By now, formal verification has been applied to many designs, algorithms, and software for critical systems (e.g., [3, 4, 5]), so this invites the question: "what is the difference between a formalized safety case and a formal verification?" In my opinion, the answer has two parts.

First, formal verification is generally concerned with *correctness*, whereas a safety case is concerned with *perfection*. Correctness is the more limited notion; for software systems, it is usually a demonstration that the high-level software requirements are properly implemented by the corresponding programming language source code, which may be in C, SCADE, or Stateflow/Simulink, etc. Perfection, on the other hand, is an all-encompassing notion: it asks whether the high-level software requirements are themselves correct (relative to the more abstract system requirements), and whether the software as it runs (with the support of the relevant compilers, linkers, middleware, operating system, and hardware) really has the behavior assumed by the semantics of the programming languages concerned, and satisfies the safety objectives identified in the system requirements. The crucial relationship are that any violation of the critical claims is a *failure*, its cause is a *fault*, and perfect software is that which has no faults. Thus, a perfect system is one that will never suffer a (safety-relevant) failure. This use of the term "perfection" comes from literature on the nature of software assurance and the statistical basis for certification [6].

In many industries, assurance of software correctness is performed and assessed by different groups and according to different guidelines than assurance of perfection for the subsystem of which it is a part. In commercial aircraft, for example, the system-level requirements are developed and assessed (typically by systems engineers) relative to the guidelines known as ARP 4761 [7] and ARP

4754A [8], whereas software is developed and assessed (typically by software engineers) relative to the guidelines known as DO-178C [1]. One of the top-level "objectives" of DO-178C is to establish that the high-level software requirements are consistent with the system-level requirement; thereafter, DO-178C is about correctness, all the way down to the running program.

The second part of the answer to the difference between a formal verification and a safety case is that a formal verification generally takes the formalized premises and conclusion as given and is focused on the demonstration that the latter follows from the former. A safety case will add to this careful justification that the premises are true of the system concerned, and that the formal conclusion does indeed support the real-world interpretation required of it. An instructive illustration of these differences between a formal verification and a "case" is provided by formal verification of the Ontological Argument [9]: this is a formally correct proof of the existence of God—a claim that will cause most readers to examine its premises and the formalization of its conclusion with especial interest.

The two parts to this answer are different sides of the same coin: a safety case tackles a bigger problem (i.e., perfection) than verification (resp. correctness), so there is more to do. It may seem that what is being proposed here is to enlarge the scope of formal verification from correctness to perfection. But, of course, those topics that traditionally are excluded from formal verification and left to the larger safety case are (mostly) excluded for good reasons: generally, they do not lend themselves to formal analysis, or they concern interpretation of the formal analysis itself. So what is proposed here is not the same as expanding verification from correctness to perfection but, instead, augmenting formal verification to include the skeleton of the rest of the safety argument.

By this I mean that we wish to use formal methods to keep track of safety case claims, and what follows from what, and why, but we do not expect to reduce those down to self-evident axioms and formalized theories. For example, to record why some formally verified theorem is considered to discharge an informally stated safety claim we may employ a proof rule equivalent to "because an expert says so," with the expert's informal justification attached as a comment. For another example, the top-level of the safety case might be organized as an enumeration over hazards; this organization could be justified by reference to a collection of patterns for safety-case arguments, while the list of hazards is justified by another "because an expert says so" proof rule, with the process of hazard discovery attached as a comment. This approach to formalization of safety cases is developed in more detail in a previous paper [10]. The idea is that the full resources of automated deduction within a formal verification system become available for the purpose of evaluating the argument of a safety case. The description in [10] envisaged augmenting a formal verification system to support this activity (chiefly by providing ways to manage the narrative comments justifying informal proof steps), but a recent alternative is to use a framework such as the "Evidential Tool Bus" (ETB) [11], which is designed to assemble formal claims from multiple sources.

My suggestion is that an approach like this could largely eliminate "logic doubt" as a concern when evaluating a safety case. Evaluators would have not only the soundness guarantee of the relevant verification system or ETB, but could actively probe the argument using "what-if" exploration (e.g., temporarily remove or change an assumption and observe how the proof fails, or inspect a counterexample). Active exploration touches on the communication aspect of a safety case and in this regard it is worth noting that proponents of safety cases sometimes recommend Toulmin's approach to framing arguments [12] rather than traditional logic [13]. Toulmin stresses justification rather than inference and adds "qualifier," "backing," and "rebuttal" components to the traditional "warrant" (i.e., proof) in presenting the derivation of a claim from evidence. Toulmin was writing in the 1950s, when mechanized support for argument was barely more practicable than it was in Leibniz' day and, in my opinion, his approach reflects the limitations of the technology at his disposal (basically, the printed page), where the arguer must attempt to anticipate and prepare responses to objections and challenges by those he would persuade but will not be able to interact with in real time. Nowadays, automated deduction does allow real-time interaction and I believe this provides a better vehicle for communication and the exploration of logic doubt than Toulmin's approach.

3 Epistemic Doubt

If we now contemplate the tasks facing an evaluator who must appraise a formalized safety case of the kind advocated above, we see that "logic doubt" is largely eliminated and concern will instead focus on the leaves of the argument. In a conventional formal verification, these would be the formal models and axioms comprising the premises of the verification, together with questions about their interpretation and that of the conclusion; in the expanded treatment that formalizes the safety case, the leaves will also include informal expert justification. I claim that all of this leaf-level material is *epistemic* in nature: that is to say, it concerns our knowledge about the system and its place in the world, including its context, requirements, environment, design, construction, hazards, and everything else that is germane to its safety.

The claim in the previous paragraph makes explicit an observation that I believe has long been accepted implicitly: there are just two kinds of concern underlying system safety, and other similar kinds of system analysis: those that are epistemic in nature (i.e., concerning our knowledge) and those that are about logic (i.e., our reasoning, based on that knowledge). As evidence, I cite the traditional partitioning of system assurance into verification and validation: the former ("did I build the system right?") is about logic, while the latter ("did I build the right system?") is about epistemology.

If mechanically supported formal reasoning allows us to reduce logic doubts, then the remaining opportunity for improving the reasoning aspect of safety cases is to reduce epistemic doubt. This means that large informal justifications attached as comments to proof rules of the flavor "because an expert says so" should be broken into more manageable pieces connected by explicit reasoning.

Furthermore, we should strive to find ways to represent expert knowledge in ways that support the communication aspect of the case while, at the same time, being directly useful in the reasoning aspect. In essence, this means we should represent our knowledge in logic. Software *is* logic, so there is, in principle, no obstacle to representing its epistemology (requirements, specification, code, semantics) in logic: that is why formal verification is feasible—and increasingly practical and cost-effective—for software.

The world with which the software interacts—the world of devices, machines, people and institutions—is not (outside of analytic philosophy) typically considered a manifestation of applied logic, but I believe there are indications that it can be. It is increasingly common that system developers build models of the world using simulation environments such as Simulink/Stateflow. These models represent their epistemology, which they refine and validate by conducting simulation experiments. Other simulation environments, developed for human-computer-interaction (HCI) studies, model aspects of human behavior as well as machines [14, 15], and models of larger human organizations are employed in many fields ranging from disaster planning to economics and politics.

The simulation environments that support these modeling activities are themselves software and they could, with difficulty, be represented within a logic-based assurance framework (this is already feasible for Simulink, whose models can be imported into many verification environments [16]). However, this is not the main obstacle to the use of simulation models within formalized assurance cases. Rather, the problem is that simulation models are designed for that purpose and simultaneously say too much and too little for the purposes of assurance and minimization of epistemic doubt. For example, the Simulink model for a car braking system will provide equations that allow calculation of the exact rate of deceleration in given circumstances (which is more information than we need), but will not provide (other than indirectly) the maximum stopping distance—which is an example of a property that may be needed in an assurance case. The crucial point is that it should be easier to resolve epistemic doubts about a simple constraint, such as maximum stopping distance, than the detailed equations that underlie a full simulation model.

So my proposal is that for the purpose of recording the epistemology of a safety case, models should be expressed as systems of constraints rather than as simulation models: less is more. Until fairly recently, it would have been difficult to validate systems of constraints: unlike simulation models, it was not feasible to run experimental calculations to check the predictions of the model against intuition and reality. Fortunately, we now have technology such as "infinite bounded model checkers," based on highly effective constraint solvers for "satisfiability modulo theories" (SMT) that allow exploration of constraint-based models (see [17, 18] for some simple examples).

I believe that constraint-based models of this kind could be particularly useful in validating system-level requirements. All software-related incidents in commercial aircraft have their origin in imperfect system-level requirements, or in mismatches between the system-level requirements and top-level software re-

quirements [19], both of which may be due, in part, to lack of good notations and tools for representing and validating system knowledge at this level of abstraction.

4 Research Directions

The diagnosis and proposals in this paper are deliberately speculative and, perhaps, provocative. The basic claim is that evaluation of the reasoning element of large safety cases will benefit from—indeed, requires—automated assistance: as Leibniz said, rather than contemplation and discussion, "let us calculate." I argued that there are just two components, and hence two sources of doubt, in a safety case: our epistemology (what we know about the system, its context, and its environment) and our logic (the validity of our reasoning about the safety of the system, given our knowledge). Formal verification systems provide tools that can be adapted to represent, analyze, and explore the logic of our case, thereby largely eliminating logic doubt, so that the tougher challenge is to represent our epistemology in a form that can be used by such systems. This can be accomplished by building models in logic that describe the elements of our knowledge (the behavior of the environment etc.); these models should be described as systems of constraints (rather than, say, simulation models) and these can be explored and validated using modern tools based on SMT solvers.

Suggested research directions are simple: try this out and see if it works. I suspect that it will be much more difficult to find ways to justify adequate completeness of our epistemology (e.g., that we have identified *all* hazards) than its validity, and that this will pose a challenging research problem. Another challenge is to reconcile the communication and reasoning aspects of a safety case that is represented in logic; preliminary ideas on this topic are presented in [20].

Acknowledgements. This work was supported by NASA under contract NNA13AB02C with Drexel University and by the DARPA HACMS program under contract FA8750-12-C-0284 with AFRL. The content is solely the responsibility of the author and does not necessarily represent the official views of NASA or DARPA.

References

1. Requirements and Technical Concepts for Aviation (RTCA) Washington, DC: DO-178C: Software Considerations in Airborne Systems and Equipment Certification. (2011)
2. Haddon-Cave, C.: 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 (2009).
3. Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., et al.: seL4: Formal verification of an OS kernel. In: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, ACM (2009) 207–220

4. Miner, P., Geser, A., Pike, L., Maddalon, J.: A unified fault-tolerance protocol. In: Formal Techniques in Real-Time and Fault-Tolerant Systems. LNCS Volume 3253, Grenoble, France, Springer-Verlag (2004) 167–182
5. Narkawicz, A., Muñoz, C.: Formal verification of conflict detection algorithms for arbitrary trajectories. Reliable Computing **17** (2012) 209–237
6. Littlewood, B., Rushby, J.: Reasoning about the reliability of diverse two-channel systems in which one channel is "possibly perfect". IEEE Transactions on Software Engineering **38** (2012) 1178–1194
7. Society of Automotive Engineers: Aerospace Recommended Practice (ARP) 4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment. (1996)
8. Society of Automotive Engineers: Aerospace Recommended Practice (ARP) 4754: Certification Considerations for Highly-Integrated or Complex Aircraft Systems. (1996) Also issued as EUROCAE ED-79; revised as ARP 4754A, December 2010.
9. Rushby, J.: The Ontological Argument in PVS. In Shilov, N., ed.: Fun With Formal Methods, St Petersburg, Russia (2013) Workshop in association with CAV'13.
10. Rushby, J.: Formalism in safety cases. In Dale, C., Anderson, T., eds.: Making Systems Safer: Proceedings of the Eighteenth Safety-Critical Systems Symposium, Bristol, UK, Springer (2010) 3–17.
11. Cruanes, S., Hamon, G., Owre, S., Shankar, N.: Tool integration with the Evidential Tool Bus. In Giacobazzi, R., Berdine, J., Mastroeni, I., eds.: Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013. LNCS Volume 7737, Rome, Italy, Springer-Verlag (2013) 275–294
12. Toulmin, S.E.: The Uses of Argument. Cambridge University Press (2003) Updated edition (the original is dated 1958).
13. Bishop, P., Bloomfield, R., Guerra, S.: The future of goal-based assurance cases. In: DSN Workshop on Assurance Cases: Best Practices, Possible Obstacles, and Future Opportunities, Florence, Italy (2004)
14. Pritchett, A.R., Feigh, K.M., Kim, S.Y., Kannan, S.: Work Models that Compute to support the design of multi-agent socio-technical systems. IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans (under review)
15. Bolton, M.L., Bass, E.J.: Evaluating human-automation interaction using task analytic behavior models, strategic knowledge-based erroneous human behavior generation, and model checking. In: IEEE International Conference on Systems, Man, and Cybernetics, Anchorage, AK (2011) 1788–1794
16. Miller, S.P., Whalen, M.W., Cofer, D.D.: Software model checking takes off. Communications of the ACM **53** (2010) 58–64
17. Rushby, J.: A safety-case approach for certifying adaptive systems. In: AIAA Infotech@Aerospace Conference, Seattle, WA, American Institute of Aeronautics and Astronautics (2009) AIAA paper 2009-1992.
18. Bass, E.J., Feigh, K.M., Gunter, E., Rushby, J.: Formal modeling and analysis for interactive hybrid systems. In: Fourth International Workshop on Formal Methods for Interactive Systems: FMIS 2011. Volume 45 of Electronic Communications of the EASST., Limerick, Ireland (2011)
19. Rushby, J.: New challenges in certification for aircraft software. In Baruah, S., Fischmeister, S., eds.: Proceedings of the Ninth ACM International Conference On Embedded Software: EMSOFT, Taipei, Taiwan, Association for Computing Machinery (2011) 211–218
20. Rushby, J.: Mechanized support for assurance case argumentation. In: Proceedings of 1st International Workshop on Argument for Agreement and Assurance (AAA 2013). Kanagawa, Japan, Springer-Verlag LNCS (2013) To appear.