Presented at the 1st International Workshop on Argument for Agreement and Assurance (AAA 2013), Kanagawa Japan, October 2013. Appears in Springer LNAI Vol. 8417, pp. 304–318.

Mechanized Support For Assurance Case Argumentation

John Rushby

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

Abstract. An assurance case provides an argument that certain claims (usually concerning safety or other critical properties) are justified, based on given evidence concerning the context, design, and implementation of a system. An assurance case serves two purposes: reasoning and communication. For the first, the argument in the case should approach the standards of mathematical proof (though it may be grounded on premises—i.e., evidence—that are equivocal); for the second it must assist human stakeholders to grasp the essence of the case, to explore its details, and to challenge it. Because of the scale and complexity of assurance cases, both purposes benefit from mechanized assistance. We propose simple ways in which an assurance case, formalized in a mechanized verification system to support the first purpose, can be adapted to serve the second.

1 Introduction

An assurance case provides an *argument* that certain *claims* about a system (usually concerning safety or other critical properties) are justified, based on *evidence* concerning its context, design, and implementation [1, 2].

The assurance case for a real system is a massive artifact: typically thousands of pages of documentation, diagrams, analyses, and tests. It is surely difficult to evaluate the argument that binds such a large amount of evidence together and connects it to the claims. Greenwell and colleagues examined three industrial safety cases and discovered logical fallacies in all of them [3]. Furthermore, each case was examined by two reviewers and there were considerable differences in the flaws detected by each reviewer.

Thus, it seems that human review is not particularly reliable for assurance case arguments and that mechanized support could add precision to their construction and analysis. Modern formal verification systems (such as Acl2, Agda, Coq, Isabelle, or PVS) provide notations adequate to the formalization and specification of complex systems and the automation (based on theorem proving and model checking) to analyze them (see, e.g., [4–6]). As we will explain, verification is a narrower problem than system assurance, but it seems plausible that the application of formal verification systems might be extended from verification

to the analysis of assurance case arguments, and there are proposals for doing this [7–9].

These proposals presuppose that the argument of an assurance case should be deductively sound, but there are differing views on this. Some believe such arguments are quintessentially inductive: they provide strong evidence that the conclusion is highly probable, not proof that it is certain [10]. My view is that we may have doubts about some of the premises (i.e., evidence) used in the argument, but that the reasoning, *given these premises*, should be logically or deductively sound. I call this the *reasoning aspect* of the assurance case and argue that formal verification methods can eliminate *logic doubt* concerning this aspect of the argument, allowing attention to be focused on *epistemic doubt* about the accuracy and completeness of our knowledge of the system, as represented in the premises to the argument [11].

But, whereas logic doubt can be eliminated by mechanized verification, epistemic doubt requires human review. There is much evidence that human individuals and groups are prone to confirmation bias, so reviews should actively challenge and explore the assumptions and claimed knowledge underlying a case. Prior to, and in addition to, review, a case also serves as a vehicle for communication and shared understanding among its stakeholders and these purposes, too, are likely to be best served by active exploration and "what-if" inquiry, rather than passive appraisal. Thus, the reasoning aspect of a case is complemented by a *communication aspect* that focuses on exploration of its epistemic foundation.

In support of its communication aspect, the epistemic foundation of an assurance case will be explored, modified, and revised—possibly many times—and its reasoning aspect will be adjusted correspondingly. Thus, seen in the large, the reasoning in an assurance case, although it can be supported locally by the tools of formal verification, is not a proof, but an *argument*: the distinction being that an argument is *defeasible*—i.e., it is subject to revision in the light of objections or new information, or for the purposes of exploration. Just as the reasoning or deductive aspect of an assurance case can benefit from mechanized support, so can its communication or defeasible aspect—and, of course, the mechanized support for each aspect must somehow coexist with the other.

In the following section, I review some topics in applying mechanized support to the reasoning and the communication aspects of an assurance case, respectively. Then, in Section 3, I propose a simple way in which a formalized assurance case, whose mechanization is primarily intended to support reasoning, can be augmented to allow defeasibility and used to support the communication aspect also. The proposal is illustrated with a simple example. Section 4 provides brief conclusions and suggestions for further research.

2 Mechanized Support for Assurance Cases

Assurance cases are large and complex artifacts and so it is necessary to have automated support for managing the overall structure of a case, and for providing representations in graphical notations such as GSN [12] to aid comprehension.

Several such tools exist [13,14], and there are emerging standards to assist their interoperation [15,16].

However, my focus here is on mechanized support for the logical aspects of an assurance case. In the next subsection, I review some topics in applying mechanized support to the reasoning or deductive aspect of an assurance case, and in the subsection that follows I review topics in providing support for the communication or defeasible aspect of a case.

2.1 Mechanized Support for Reasoning in Assurance Cases

Modern formal methods tools such as verification systems, model checkers, and SMT solvers have sufficient expressiveness and automation to undertake the task of providing mechanized support to the reasoning or deductive aspect of an assurance case. But just as its deductive aspect is not the whole purpose of an assurance case, so classical formal verification is not quite the same as the deductive part of an assurance case, so some care is needed in the way in which an assurance case is represented as a formal verification.

A formal verification differs from the deductive aspect of an assurance case in that verification takes the specification (i.e., premises) as given and verifies correctness of the conclusions that are derived from it, whereas an assurance case must also justify (often by citing the evidence of the case) the premises and rules of inference that it uses, and it must also justify that the verified conclusion bears an interpretation that is relevant to the claim of the case. For example, in applying formal verification to an assurance case we might use a proof rule that says "a system is safe if it is shown to be safe for each of its hazards"; in applying this rule we would show safety for each hazard that has been explicitly identified and would have a premise that says these are *all* the hazards. Justification of this premise would be a major part of the safety case, yet is outside the formal verification. In [7], I proposed a way in which an assurance case can be represented in a formal verification system so that these aspects are at least recorded. In essence, I conjoin to each premise a predicate that is set *true* only when a reviewer accepts its supporting evidence, which can be attached to the predicate as a comment. A formal verification system such as PVS requires small enhancements to fully support this proposal (basically, support for referencing documents as comments), but recent tool-integration frameworks such as the Evidential Tool Bus (ETB) [17] allow nonformal justifications to be attached to claims as a basic capability.

The premises to an assurance case represented within a formal verification system record our knowledge about the system or, to use a fancier term, its *epistemology*. The soundness of the deductive aspect of an assurance case rests on two pillars: how complete and accurate is our knowledge about the system, and how accurate is our reasoning about the case, given our knowledge. Concern about the second of these (*logic doubt*) is largely eliminated by the soundness guarantee of formal verification, so concern should mainly focus on the first item (*epistemic doubt*), especially its completeness.

As suggested above, justification for many of the premises in a formalized assurance case will be references to the evidence of the case. In developing a formalized assurance case we can choose how abstractly to represent the system and, in consequence, the granularity of the evidence that is explicitly represented. Since evidence is opaque to the formal analysis, there is much to be said for refining the level of abstraction and breaking large items of evidence into more tightly focused pieces connected by explicit reasoning. 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—has not traditionally been represented in logic, but indirectly it is becoming so, for 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.

It is feasible to import models from simulation environments such as Stateflow/Simulink into verification environments (see, e.g., [18]). However, simulation models are not the best representation of the epistemology for an assurance case. 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.

A proposal, developed in [11], 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. 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. But now we have technology such as "infinite bounded model checkers," based on highly effective constraint solvers for "satisfiability modulo theories" (SMT) that allow effective exploration of constraint-based models [19].

2.2 Mechanized Support for Communication in Assurance Cases

As noted earlier, a formal verification is not the same as an assurance case and, even with the adjustments proposed above, there are purposes served by an assurance case that are not supported by its embedding in a formal verification system, as currently envisaged.

In particular, an assurance case is not purely about deductive reasoning: it is also about communication and, ultimately, persuasion. That is, an assurance case is constructed by humans and embodies their understanding and beliefs about the system and these need to be communicated to other stakeholders, including regulators and certifiers. Effective communication is unlikely to be one-way; it is more likely to be a dialog and the process of developing a common understanding may lead to revisions in the assurance case. The revisions may adjust some of the premises at the bottom of the argument, and they may adjust some of the reasoning expressed in its rules or axioms.[1]

In addition to revisions that represent adjustments to the argument, reviewers of the case may wish to temporarily change elements of the argument (i.e., conduct "what if" experiments) to assist their comprehension of the case. These permanent and temporary revisions to an assurance case suggest that its argument should be viewed as provisional, or contingent, and should therefore be developed in a framework that supports such "defeasible" reasoning.

Defeasible reasoning is well-studied in philosophy and in AI (where it is generally referred to as nonmonotonic reasoning), and there are rich bodies of work on belief revision, commonsense reasoning, truth maintenance, and so on. Closely related are the fields of reasoning under uncertainty, where we find fuzzy logic, Dempster-Shafer belief functions and so on, and probabilistic methods, where we find probabilistic and Markov logics, Bayesian Belief Networks (BBNs) and so on.[2] The field of Argumentation frames similar issues in a (generally) more abstract setting [21] where different agents may employ different sets of premises so that a premise of one may "defeat" that of another, and entire arguments may "attack" one another.

Most of these methods for defeasible reasoning, and the tools that support them, are framed as augmentations to propositional logic, whereas we earlier made the case that mechanized support for deductive reasoning in assurance cases should build on the much more powerful logics and theories of modern verification systems, model checkers, and SMT solvers.

One could imagine a two-pronged approach to mechanized support for development and evaluation of assurance cases: a powerful deductive system for analyzing the reasoning in detail, and a defeasible or argumentation-based system to support exploration and experiment on the overall argument at an abstract level for the purposes of understanding and communication. Such an approach could be viable, and it might even be possible to automate abstraction from the deductive to the defeasible levels of detail (though the reverse might be more difficult), but I believe there could be benefits in augmenting the representation

[1] There are persuasive claims that human consciousness evolved to enable communication and cooperative behavior, and that reasoning evolved to evaluate the epistemic claims of others [20]. Thus, argument is a fundamental human capability, constructive reasoning is an epiphenomenon, and confirmation bias is intrinsic.

[2] I prefer not to cite specific works from the vast repertoire of articles and books on these topics; an Internet search will provide many good references.

and tools proposed for deductive analysis of assurance cases so that they can also support the defeasible level. This is the topic of the following section.

3 Supporting Defeasible Reasoning in Mechanized Verification

In defeasible reasoning, we may draw a conclusion based on a state of knowledge that is subsequently revised, invalidating the previous conclusion. A standard example is

(1) Tweety is a bird,
(2) Birds can fly,
(3) Therefore Tweety can fly.

Subsequently, we learn that Tweety is a penguin and penguins cannot fly.

This new information contradicts our prior knowledge and there are many proposals how to adjust our logic and our reasoning to accommodate such revisions. Often, we will have both a general rule "birds can fly" and a revision "penguins cannot fly" that each apply to Tweety, who is both a bird and a penguin, and we need some method (such as "circumscription" [22]) for resolving the apparent inconsistency and preferring one conclusion over another. In other cases, a revision may flatly deny some prior rule (e.g., Tweety is not a bird but a bat) and defeasible reasoning provides ways to handle these inconsistencies, too.

While this kind of sophistication is valuable when representing commonsense reasoning, or when resolving arguments where different parties advance different premises, I do not believe it is necessary or desirable in the evaluation of assurance cases. In evaluating an assurance case about Tweety, we would wish to be alerted to the potential inconsistency in our epistemology concerning his ability to fly, but would surely then seek to reach consensus on the point and then reason classically from there. That is to say, rather than rely on logics for default or defeasible reasoning to cope with inconsistencies resulting from different opinions or conflicting evidence, we would revise our assurance case to resolve or eliminate the inconsistencies so that classical deductive reasoning provides a single conclusion (this is similar to Pollock's notion of a "warrant" [23]).

I propose that one simple way to allow exploration and challenge while still using classical reasoning is to introduce explicit "defeater" predicates into the premises of an assurance case.[3] Then, our premises concerning Tweety become

(1) Absent a defeater about Tweety, Tweety is a bird, and
(2) Absent a defeater about flying, birds can fly.

More formally, any premise p becomes $\neg d_p \supset p$ where d_p denotes the defeater for p (and $\neg$ and $\supset$ are logical negation and implication, respectively). Initially, all defeaters are absent (i.e., false) and we conclude that Tweety can fly. A reviewer who has doubts about the universality of the premise "birds can fly"

[3] Some treatments of defeasible reasoning distinguish "undercutting," "undermining," and "rebutting" defeaters, but the distinctions are not sharp and are not used here.

may turn on its defeater, observe the consequences, and revise the argument by adding additional premises and constraints about penguins. I provide an example below where the consequences of a defeater are a little less obvious, and the benefits more significant.

In addition to turning defeaters on and off and then reasoning "forwards" to deduce the consequences, we could instead assert that Tweety is a bird but cannot fly, and then reason "backwards" to seek an explanation. Observe that this is exactly the basis for model-based diagnosis [24], where our "defeater" predicates take the role of the "abnormal" predicates used in diagnosis (and the related "reconfig" predicates used in model-based repair [25]). Some of the tools that underlie modern model checkers and formal verification systems provide capabilities that directly support this kind of examination. For example, our Yices SMT solver [26] not only can generate counterexamples as well as verify large formulas in a rich combination of theories (i.e., it does SAT as well as UNSAT for SMT), but it can also generate UNSAT Cores, and perform Weighted MAXSAT for SMT.

In the following section, I illustrate the use of explicit "defeater" predicates, and also some of the other points made earlier, in a simple example.

3.1 Example

I illustrate the proposal above using a small example from [27]. Below, I reproduce the "structured prose" rendition of the assurance case from that example, to which I have added paragraph numbers.

(1) This argument establishes the following **claim**: the control system is acceptably safe, within the context of a definition of acceptably safe. To establish the top-level claim, two **sub-claims** are established: (a) all identified hazards have been eliminated or sufficiently mitigated and (b) the software has been developed to the integrity levels appropriate to the hazards involved.

(2) Within the **context** of the tolerability targets for hazards (from reference Z) and the list of hazards identified from the functional hazard analysis (from reference Y), we follow the **strategy** of arguing over all three of the identified hazards (H1, H2, and H3) to establish sub-claim 1, yielding three additional **claims**: H1 has been eliminated; H2 has been sufficiently mitigated; and H3 has been sufficiently mitigated.

(3) The **evidence** that H1 has been eliminated is formal verification.

(4) The **evidence** that catastrophic hazard H2 has been sufficiently mitigated is a fault tree analysis showing that its probability of occurrence is less than 1×10^{-6} per annum. The **justification** for using this evidence is that the acceptable probability in our environment for a catastrophic hazard is 1×10^{-6} per annum.

(5) The **evidence** that the major hazard H3 has been sufficiently mitigated is a fault tree analysis showing that its probability of occurrence is less than 1×10^{-3} per annum. The **justification** for using this evidence is that the acceptable probability in our environment for a major hazard is 1×10^{-3} per annum.

(6) We establish sub-claim (b) within the **context** of the list of hazards identified from the functional hazard analysis in reference Y, and the integrity level (IL) process guidelines defined in reference X. The process **evidence** shows that the primary

protection system was developed to the required IL 4. The process **evidence** also shows that the secondary protection system was developed to the required IL 2.

I present a few highlights from a formalization of this argument in PVS [28,29].

As soon as we start to formalize the argument, we recognize that paragraph (6) is not well connected to the rest of the case. This illustrates one of the benefits in applying mechanized checking to an assurance case: we are forced to ensure that the argument "connects up" and is deductively sound. Presumably, a more fully developed version of the argument would say that part of the fault tree analysis cited in (4) is an assumption that the software of the primary protection system has a failure rate below some threshold, and development to Integrity Level 4 (`IL4`) is considered to ensure that. Similarly for paragraph (5) and development of the secondary protection system to `IL2`.

To formalize this in PVS, we introduce the enumerated types `hazlevels` and `intlevels` to represent hazard and integrity levels respectively, and we provide axioms asserting that hazard `H2` is `catastrophic` and that process evidence attests that the system was developed to integrity level `IL4` with respect to this hazard, and similarly for hazard `H3` (we omit specifications for the signatures of the functions `hazlev` and `process`).

```
hazlevels: TYPE = { minor, major, catastrophic }
intlevels: TYPE = { IL2, IL4 }

H2hlev: POSTULATE hazlev(system, H2) = catastrophic
H3hlev: POSTULATE hazlev(system, H3) = major

H2ilev: POSTULATE process(system, H2) = IL4
H3ilev: POSTULATE process(system, H3) = IL2
```

We use the keyword `POSTULATE` to indicate premises justified by evidence; in contrast the keyword `AXIOM` indicates premises that represent the reasoning or "proof rules" employed. PVS treats these keywords as synonyms, but the distinction is useful for communication with human readers.

Next, we provide the "proof rule" axiom `pr` that relates hazard and integrity levels to the claim that a given hazard is adequately "handled." Here `sy` and `hz` are variables ranging over systems and hazards, respectively.

```
pr: AXIOM
  (hazlev(sy, hz) = catastrophic AND process(sy, hz) = IL4
OR hazlev(sy, hz) = major AND process(sy, hz) = IL2)
   => handles(sy, hz)
```

From these we can prove the lemmas that hazards `H2` and `H3` are adequately handled. There is a similar (omitted) treatment for `H1` on the basis that it has been formally verified.

```
H1OK: LEMMA handles(system, H1)
H2OK: LEMMA handles(system, H2)
H3OK: LEMMA handles(system, H3)
```

We then assert that H1, H2, and H3 are *all* the hazards for this system and claim in this context

```
H1, H2, H3: hazards

hazard_ax: POSTULATE
  allhazards(claim, system, context) = {: H1, H2, H3 :}
```

We employ the argument strategy that a system is safe if each of its hazards is adequately handled. Here cl and co are variables ranging over claims and contexts, respectively.

```
strategy: AXIOM
  LET hset = allhazards(cl, sy, co) IN
     (FORALL (h: (hset)): handles(sy, h))
        IMPLIES safe(cl, sy, co)
```

With these specifications, we can easily prove that the system is safe.

```
sysOK: THEOREM safe(claim, system, context)
```

Skeptical reviewers who examine this formalized assurance case might suggest that the level of abstraction is too high: they might be concerned about independence of H2 and H3 and be disappointed that the fault tree analyses are opaque items of evidence.[4] This illustrates the point made in Section 2.1 concerning epistemic doubt and the granularity of evidence.

There are two plausible approaches at this juncture: one is to elaborate the formalized case to include the top levels of the fault tree analyses so that the crucial topic of independence is exposed in the formal representation of the case; the other is to introduce a new hazard H23 that represents joint occurrence of H2 and H3. We will pursue the latter course here.

The developers of the assurance case might then introduce a premise that states that the joint hazard H23 is catastrophic and must be mitigated to a probability of occurrence less than 1×10^{-6} per annum, and claim that this is ensured by the combination of process evidence of IL4 for the primary system and IL2 for the secondary. The relevant changes are shown below and the formal verification of the case succeeds as before.

```
hazard_ax: POSTULATE
  allhazards(theclaim, system, context) = {: H1, H2, H3, H23 :}

H23hlev: POSTULATE hazlev(system, H23) = catastrophic

H23pr: AXIOM
  handles(system, H2) AND handles(system, H3)
    => handles(system, H23)

H23OK: LEMMA handles(system, H23)
```

[4] The prose description in [27] suggests that the system under consideration has a primary and a secondary protection system; a standard concern in these kinds of system is that both protection systems fail on the same demand [30].

Reviewers might be skeptical that the conjunction of process evidence for the primary and secondary systems, each considered in isolation, is sufficient to ensure mitigation of the *joint* occurrence represented by `H23`. To explore this they could turn on the defeater `dfH23pr` for premise `H23pr`. (To keep the presentation simple, I have not included the defeaters until now.)

```
dfH23pr: boolean = TRUE

H23pr: AXIOM NOT dfH23pr =>
  handles(system, H2) and handles(system, H3)
    => handles(system, H23)
```

The formal verification now fails to guarantee safety.

The developers of the case could then introduce new evidence that the combined primary and secondary system has been used previously in a different, but similar context (with a system called `otherS` and hazard `otherH`).

```
previous: POSTULATE
   similar((otherS, otherH),(system, H23))
    AND handles(otherS, otherH)
```

They assert this is sufficient to claim that the present system handles `H23`.

```
dfprior23: boolean = FALSE

prior23: AXIOM NOT dfprior23 =>
  similar((otherS, otherH), (system, H23)) AND handles(otherS, otherH)
   => handles(system, H23)
```

They turn off the defeater for this new premise and are once again able to verify safety.

We now have two ways to justify safety of the system: one citing evidence of integrity levels and fault tree analyses, and another citing prior experience. In a conventional formal verification there is little purpose in such redundancy of argument, but in an assurance case it can be useful. Here, the reviewers might be skeptical of the evidence by prior experience because they are uncertain that the context of the previous system is sufficiently similar to the present one, and so they turn on the defeater for this argument.

```
dfprior23: boolean = TRUE
```

Once again, the formal verification fails to guarantee safety. But now the developers might argue that although both lines of safety justification have their flaws, in combination they constitute a "multi-legged" case (with independent legs) that is surely sufficient. The reviewers might accept this and can adjust their intervention in the formalized assurance case to state that either defeater may be true, but not both together.

```
dfH23pr, dfprior23: boolean

notboth: AXIOM NOT (dfH23pr AND dfprior23)
```

Now the formal verification succeeds once again, and the reviewers are satisfied.

Here, "inspection" was sufficient to see that our epistemic foundation remained consistent as we introduced new premises and toggled defeaters, but in larger examples it will be important to use mechanized assistance to ensure this.

That concludes our small example. Its purpose was to illustrate the idea, but its small size means that it cannot illustrate what I believe is the main attraction in this approach: namely, that it can exploit the full power of modern formal methods tools and should therefore scale to large examples that use rich logics and theories.

4 Discussion and Conclusion

This paper has reviewed some topics in providing mechanized support for the analysis and exploration of arguments in assurance cases. We saw that powerful modern tools for formal methods can provide useful support for the deductive or reasoning aspect of an assurance case and we explored some of the issues in representing cases so that epistemic doubts are minimized. We then considered support for the communication aspect of assurance cases and concluded that this requires some element of defeasible reasoning. However, we suggested that the purposes served by assurance cases are such that special logics for defeasible reasoning or abstract argumentation are unnecessary—in fact, undesirable—and that adequate support can be obtained by simply adding explicit "defeater" predicates to the premises of the formalized case. We illustrated this with a simple example. Related work includes similar proposals by Kinoshita and Takeyama [31].

Notice that our defeater predicates are not the same as the defeaters of Pollock [23,32], where defeaters are premises that contradict other premises and some mechanism is required to derive a preferred conclusion in the face of these inconsistencies. Our defeater predicates are used to turn premises on or off so that classical reasoning can be used. It is therefore important to check, for any given assignment of values to defeater predicates, that the enabled premises are not contradictory; unlike in our example, this check should be automated. Notice also that our defeater predicates are either given explicit truth assignments or the conclusions to the verification are true under all interpretations (possibly subject to constraints, such as `notboth` in the example) so, contrary to [33, section 6.1], philosophical objections to "logically-uninterpreted conditions" do not apply.

Future research could include comparison with proposals that do employ more sophisticated treatments of defeasible argumentation, such as [32,34]. Some treatments of argumentation, beginning with Dung [35], relate this to logic programming, and it would be interesting to explore the extent to which this can be supported in the Evidential Tool Bus, where the underlying framework is Datalog [17].

The comparison with argumentation frameworks should consider philosophy as well as technology. Argumentation generally presupposes a context where participants have different points of view and there may be no single "correct" conclusion (for example, arguments about ethics or aesthetics), or where participants have limited access to ground truth (e.g., drawing conclusions on the basis

of imperfect sensors). Argumentation methods will evaluate proffered arguments and their defeaters or their attack relations and will derive conclusions, but these may not be deductively sound. In contrast, I believe that while argumentation may be an appropriate framework during development of an assurance case, the finished case should be one in which every credible objection has been anticipated and incorporated into the argument in such a way that the conclusion is deductively sound. Exploration and examination of the case then focuses on epistemic doubt about the premises, aided by the presence of defeater predicates that enable what-if experimentation.

Related to the philosophy and the purpose of an assurance case, Steele and Knight [36] provide a very illuminating account of certification, which I formulate as follows. The system under consideration is a designed artifact and may have flaws that could lead to accidents. The task of safety-critical design is to identify and either eliminate or mitigate all hazards to its safe deployment. The task of an assurance case is to provide confidence that this has been done, correctly and completely. But the assurance case itself is a designed artifact and may have flaws that could lead to a "certification accident": that is, the decision to approve and allow deployment of a potentially unsafe system. So the principles of safety-critical design should be applied "recursively" to the assurance case itself. That is, we should use systematic methods, inspired by those used for systems (e.g., fault tree analysis), actively to seek hazards (i.e., defeaters) to the assurance case, and should then seek to eliminate or mitigate them. Mitigation could take the form of a multi-legged case, as used in the example of the previous section, where an attractive method of justification could be that the defeaters of each leg are independent [37].

An excellent topic for future research is to explore the application and consequences of Steele and Knight's insight and its representation within the framework proposed here. A related topic is to explore the novel structure for assurance cases proposed by Hawkins and colleagues [10], who divide the overall case into a safety argument and a confidence argument.

A final topic for future research is to explore whether it may be feasible to derive some measure for the confidence in a case from the number and the nature of the defeaters that are accommodated. One way to do this would be to attach subjective probabilities to defeaters and then use calculation in some suitable probabilistic framework such as Bayesian Belief Networks (BBNs); another might be to employ "Baconian probabilities" as proposed in [38].

Acknowledgements. I am grateful for helpful comments by the reviewers that caused me to rethink some of the presentation, and to stimulating discussions with Michael Holloway and John Knight.

This work was supported by NASA under contracts NNA13AB02C with Drexel University and NNL13AA00B with the Boeing Company, and by SRI International. The content is solely the responsibility of the author and does not necessarily represent the official views of NASA.

References

1. Bishop, P., Bloomfield, R.: A methodology for safety case development. In: Safety-Critical Systems Symposium, Birmingham, UK (1998)
2. Kelly, T.: Arguing Safety—A Systematic Approach to Safety Case Management. PhD thesis, Department of Computer Science, University of York, UK (1998)
3. Greenwell, W.S., Knight, J.C., Holloway, C.M., Pease, J.J.: A taxonomy of fallacies in system safety arguments. In: Proceedings of the 24th International System Safety Conference, Albuquerque, NM (2006)
4. 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
5. Miner, P., Geser, A., Pike, L., Maddalon, J.: A unified fault-tolerance protocol. In: Formal Techniques in Real-Time and Fault-Tolerant Systems. Volume 3253 of Lecture Notes in Computer Science., Grenoble, France, Springer-Verlag (2004) 167–182
6. Narkawicz, A., Muñoz, C.: Formal verification of conflict detection algorithms for arbitrary trajectories. Reliable Computing **17** (2012) 209–237
7. 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
8. Basir, N., Denney, E., Fischer, B.: Deriving safety cases from automatically constructed proofs. In: 4th IET International Conference on System Safety, London, UK, The Institutions of Engineering and Technology (2009)
9. Takeyama, M., Kido, H., Kinoshita, Y.: Using a proof assistant to construct assurance cases: Correctness by construction (fast abstract). In: The International Conference on Dependable Systems and Networks, Boston, MA, IEEE Computer Society (2012)
10. Hawkins, R., Kelly, T., Knight, J., Graydon, P.: A new approach to creating clear safety arguments. In Dale, C., Anderson, T., eds.: Advances in System Safety: Proceedings of the Nineteenth Safety-Critical Systems Symposium, Southampton, UK, Springer (2011)
11. Rushby, J.: Logic and epistemology in safety cases. In: SafeComp 2013: Proceedings of the 32nd International Conference on Computer Safety, Reliability, and Security. Volume 8153 of Lecture Notes in Computer Science., Toulouse, France, Springer-Verlag (2013) 1–7
12. Spriggs, J.: GSN—The Goal Structuring Notation. Springer, London, UK (2012)
13. Denney, E., Pai, G., Pohl, J.: 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 (2012)
14. ASCE: ASCE home page http://www.adelard.com/web/hnav/ASCE/index.html.
15. SACM: OMG Structured Assurance Case Metamodel (SACM) home page http://www.omg.org/spec/SACM/.
16. MACL: OMG Machine-Checkable Assurance Case Language (MACL) home page http://www.omg.org/cgi-bin/doc?sysa/2012-9-4/.
17. 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 (VMCAI), 14th International Conference. Volume 7737 of Lecture Notes in Computer Science., Rome, Italy, Springer-Verlag (2013) 275–294

18. Miller, S.P., Whalen, M.W., Cofer, D.D.: Software model checking takes off. Communications of the ACM **53** (2010) 58–64
19. Rushby, J.: Harnessing disruptive innovation in formal verification. In Hung, D.V., Pandya, P., eds.: Fourth International Conference on Software Engineering and Formal Methods (SEFM), Pune, India, IEEE Computer Society (2006) 21–28
20. Mercier, H., Sperber, D.: Why do humans reason? arguments for an argumentative theory. Behavioural and Brain Sciences **34** (2011) 57–111 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".
21. Chesñevar, C.I., Maguitman, A.G., Loui, R.P.: Logical models of argument. ACM Computing Surveys **32** (2000) 337–383
22. McCarthy, J.: Circumscription—a form of non-monotonic reasoning. Artificial Intelligence **13** (1980)
23. Pollock, J.L.: Defeasible reasoning. Cognitive Science **11** (1987) 481–518
24. Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence **32** (1987) 57–95
25. Crow, J., Rushby, J.: Model-based reconfiguration: Toward an integration with diagnosis. In: Proceedings, AAAI-91 (Volume 2), Anaheim, CA (1991) 836–841
26. Yices: Yices home page http://yices.csl.sri.com/.
27. Holloway, C.M.: Safety case notations: Alternatives for the non-graphically inclined? In: 3rd IET International Conference on System Safety, Birmingham, UK, The Institutions of Engineering and Technology (2008)
28. Owre, S., Rushby, J., Shankar, N., von Henke, F.: Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. IEEE Transactions on Software Engineering **21** (1995) 107–125
29. PVS: PVS home page http://pvs.csl.sri.com/.
30. 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
31. Kinoshita, Y., Takeyama, M.: Assurance case as a proof in a theory: towards formulation of rebuttals. In Dale, C., Anderson, T., eds.: Assuring the Safety of Systems: Proceedings of the 21st Safety-Critical Systems Symposium, SCSC (2013) 205–230
32. Pollock, J.L.: Defeasible reasoning with variable degrees of justification. Artificial Intelligence **133** (2001) 233–282
33. Staples, M.: Critical rationalism and engineering: Ontology. Synthese (2014) To appear.
34. Caminada, M.W.A.: A formal account of Socratic-style argumentation. Journal of Applied Logic **6** (2008) 109–132
35. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artificial Intelligence **77** (1995) 321–357
36. Steele, P., Knight, J.: Analysis of critical system certication. In: 15th IEEE International Symposium on High Assurance Systems Engineering, Miami, FL (2014)
37. Goodenough, J.B., Weinstock, C.B., Klein, A.Z., Ernst, N.: Analyzing a multi-legged argument using eliminative argumentation. In: Layered Assurance Workshop, New Orleans, LA (2013)
38. Weinstock, C.B., Goodenough, J.B., Klein, A.Z.: Measuring assurance case confidence using Baconian probabilities. In: 1st International Workshop on Assurance Cases for Software-Intensive Systems (ASSURE), San Francisco, CA (2013)