Formal verification, scientific code, and the epistemological heterogeneity of computational science

Cyrille Imbert, Vincent Ardourel

DOI: 10.1017/psa.2022.78

This is a manuscript accepted for publication in *Philosophy of Science*.
This version may be subject to change during the production process.

Formal verification, scientific code, and the epistemological heterogeneity of computational science[1]

Cyrille Imbert[2] and Vincent Ardourel[3,4]

Abstract. Various errors can affect scientific code and detecting them is a central concern within computational science. Could formal verification methods, which are now available tools, be widely adopted to guarantee the general reliability of scientific code? After discussing their benefits and drawbacks, we claim that, absent significant changes as regards features like their user-friendliness and versatility, these methods are unlikely to be adopted throughout computational science, beyond certain specific contexts for which they are well-suited. This issue exemplifies the epistemological heterogeneity of computational science: profoundly different practices can be appropriate to meet the reliability challenge that rises for scientific code.

[1] This paper is dedicated to the memory of Margaret Morrison (1954–2021).

[2] Archives Poincaré, CNRS — Université de Lorraine, France. cyrille.imbert@univ-lorraine.fr.

[3] IHPST, CNRS — Université Paris 1 Panthéon-Sorbonne, France. vincent.ardourel@univ-paris1.fr.

[4] This paper is the result of work that was deeply collaborative. Both authors contributed to all stages of the research process, albeit in different proportions: CI was slightly more involved in the writing of the paper and VA in the analysis of the technical literature.

Acknowledgments. We thank the reviewers for their helpful and constructive comments, as well as Ben Young for suggestions concerning linguistic issues. All remaining shortcomings are ours.

Keywords. Computational science; Code; Software; Formal verification; Scientific practices; Epistemology of science; Heterogeneity in science; Good research practices; Standardization

1. Introduction. Various types of errors can affect scientific code, the reliability of which is a central concern for computational science (Merali 2010). While errors can sometimes be innocuous, they can also spoil scientific results and have dramatic consequences: it was a conversion error from a 64-bit floating-point number to a 16-bit signed integer value that caused the maiden flight of the Ariane 5 launcher in 1996 to end in disaster (Lions 1996, 4). Thus, the verification of scientific codes should be a top priority task in computational science, for which the most stringent and powerful tools available can be expected to be used. Surprisingly, this does not seem to be entirely the case, as the case of formal verification methods shows.

Formal verification methods are automatized procedures based on rigorous mathematical tools that aim to check the correctness of code. They can address specification issues and verification tasks in programs or design issues within computation systems (e.g., Butler and Muñoz 2016; Rushby 2007). That they are neglected (or, at least, partially so) is all the more unexpected since they no longer represent an unattainable holy grail for practitioners. Certainly, for a long time, they were more an object of inquiry for computer scientists (and of false promises) than a tool that was available for scientists; yet today things are presently different and some scientists are already making use of these methods. Nevertheless, whereas these tools seem to have the potential to boost the reliability of codes, they are not widely adopted by scientists. This disjuncture appears even more paradoxical given that scientists are, we have every reason to believe, eager to warrant the validity of their inquiries. So, from both a scientific and an epistemological perspective, it seems even more legitimate today to ask: “If this stuff is so good, why isn’t it used more?”

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

(Rushby 1997, 18). Why does the scientific community not seize on this type of practice and why does it continue to shun one of the best tools for increasing confidence in scientific code?

To answer these questions, in this paper we study the epistemological status of formal verification methods in scientific practice; we ask whether they can indeed be widely adopted in computational science to verify code; and we analyze what their partial neglect tells us about the epistemology of computational science. For this purpose, we first review philosophical discussions on verification in computational science, emphasizing that questions concerning formal verification have largely so far been ignored (section 2). Then, we present formal verification methods and describe their increasing use in computational science (section 3). We scrutinize their pros and cons, and highlight the deeply rooted obstacles to their generalized adoption, from their mathematical features to the community organization their use requires. On this basis, we argue that, absent significant changes concerning their user-friendliness, these methods are unlikely to be widely adopted as standard verification practices across the computational sciences (section 4). However, we emphasize that these demanding, high-profile methods can be well-suited to specific contexts, and we outline the profile of these contexts (section 5). Finally, we analyze the consequences of this scientific heterogeneity both from an epistemological and a methodological perspective (sections 6 and 7).

2. Verification in Computational Science. A key issue in the study of computational science is to understand how it differs more than quantitatively from non-computational

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

science and whether it raises novel epistemological questions (Frigg and Reiss 2009; Humphreys 2009). Because it involves specific types of processes, computational science also has specific ways of succeeding and failing. Since such processes are complex and have many ways of going wrong, "knowledge-based precautions against error are routine with computational methods" (Humphreys 2004 112). In this context, questions related to the analysis of errors, their sources, and how to guard against them are emerging as a specific domain of inquiry (Fresco and Primiero 2013, Beisbart and Saam 2019). Concerning this issue, philosophers have mostly focused on mathematical and modeling aspects: they have, for example, investigated how numerical errors may threaten reproducibility (Lenhard and Küster 2019), whether they can be treated as modeling errors (Fillion and Corless 2014), and what impact they have concerning "the justification of results obtained by computer simulation" (Fillion 2017, 138).

Code verification remains far less discussed by epistemologists of computational science. This topic surfaces mainly in discussions about the possibility of keeping verification and validation distinct when assessing the reliability of computer simulations. Checking the reliability of empirical inquiries that use computational methods involves various tasks: these typically include the design of one or several models representing a target system, the development of mathematical and computational techniques to explore these models, the writing of well-specified code to implement the corresponding algorithms, the running of the code on suitable machines, the collection of computational results, and the analysis of results so as to derive information about the models under investigation and the systems represented. 'Verification and validation' is a central

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

methodology for engineers, in which verification is seen as concerning the adequateness of the code and the algorithms that explore the mathematical model, while validation aims to evaluate whether the mathematical model correctly represents the target phenomenon (Oberkampf and Roy 2010). Success depends on how reliably these tasks are completed (see again Beisbart and Saam 2019 for details).

Notably, the fact that these tasks are conceptually distinct does not imply that they are separately carried out. Accordingly, the question arises of the possibility of assessing the reliability of their completion separately, by analyzing the mathematical adequateness of the target algorithms and how they are implemented on the one hand and by validating models on the other. This question has been intensively discussed over the last decade. On the one side, Winsberg maintains that "[t]he sanctioning of simulations does not cleanly divide into verification and validation" (Winsberg 2010, 23), citing miscellaneous evidence in favor of this claim, from the general inability to prove by mathematical means that verification has been achieved, to the analysis of benchmarking as a global activity, or the existence of back-and-forth or trial-and-error practices (Winsberg 2010, chap. 2). Lenhard also argues that although verification and validation "should be distinguished in a conceptual sense, they are interrelated in practice" (Lenhard 2019, 180), rooting his arguments in the analysis of practices found in software engineering. On the other side, Morrison (2015, chap. 7) argues that Winsberg overstates the impossibility of verifying computer simulations and, to this end, presents methods used by scientists, such as error assessment or the method of manufactured solutions (see also Fillion (2017) and Rider (2019) for similar discussions in the context of equation-solving through error analysis and

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

benchmarks). Beisbart (2019) suggests that confusion in this debate may arise from neglect of the distinction between conceptual and computational models. Finally, Jebeile and Ardourel (2019) defend the in-practice possibility of disentangling verification and validation by discussing the application of formal verification methods by the U.S. Army Corps of Engineers to study models of hurricane storm surges.

Formal methods of code verification are relevant to this discussion since they provide a way to verify code independently, stringently, and through abstract automated processes. Surprisingly, whereas computer scientists, software engineers, and sociologists have been discussing their use for decades (Shapiro 1997; McKenzie 2001), philosophers have almost entirely ignored them. Be this as it may, the use of formal methods and that of the entanglement of the verification and validation of computational inquiries are two issues that, though related, remain distinct. When one can use formal methods to verify computational inquiries, an important step is made toward the disentanglement of code verification and the validation of inquiries. Nevertheless, the impossibility of using formal verification methods does not imply that code verification and validation are necessarily entangled. For example, code verification may be carried out by other non-formal methods focused on code. An example is N-version programming (NVP), a method aimed at detecting errors in codes by involving different individuals or groups who code independently (see, e.g., Shapiro (1997, 34) for a discussion). In the same vein, Creel notes that an "algorithm is almost always implemented in multiple languages" (2020, 18) and may return different outputs for the same target function depending on how it is implemented. Accordingly, a means to uncover coding errors is to compare the results

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

issuing from different software packages, as was done successfully by cosmologists working on the Dark Energy Survey (ibidem, 19).

Other non-formal methods, not focused on code, can be employed to verify programs, such as the method of manufactured solutions, the use of benchmarks, or a posteriori verification methods such as backward error analysis (see, e.g., Fillion and Moir, 2018). For the sake of argumentative clarity, we will leave the issue of entanglement for future work, and we focus in this paper only on the sheer possibility of using formal methods for code verification within computational science. However, the above discussion suggests several lessons for the present project.

First, details matter: epistemologists need to go deeper into the presentation, analysis, and discussion of verification methods to discuss their scientific usefulness. Conversely, unless their scope is stringently justified, general claims on this issue are unlikely to be conclusive. Second, the above discussion concerning the possibility of verification tends to be presented as an all-or-nothing matter. But this is too crude since what is possible in practice is, for a large part, a gradual matter that depends on various contextual factors. Further, verification may bear on multiple aspects of the code, and this activity may itself be carried more or less extensively. Even if total verification proves impossible, there is much to be analyzed concerning how much partial verification of code is possible and should be promoted as a good rule of scientific conduct. Third, a more precise investigation is needed of the factors and constraints that make the use of verification methods possible or not. Such factors and constraints may be of various kinds: logical, computational, and mathematical aspects are naturally crucial, but other essential factors

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

may pertain to the development of scientific communities, how relevant knowledge and know-how are shared, or how scientific labor is divided. Fourth, the question of the appropriateness of using verification methods cannot be limited to epistemic factors. As we shall see, values can play a significant role in the resort to particular methods (see Morrison 2014 for preliminary analyses). Finally, while general claims about computational science are defended in the above-described discussions, positions are based on particular cases and practices. The extent to which those cases are representative and whether it is legitimate to oppose them is rarely questioned. This is a worry if the usability, advantages, and suitability of computational methods depends on contextual factors. Based on these insights, in the remainder of this paper we investigate specifically whether formal verification methods offer bright prospects for the whole of computational science.

3. Formal Methods and their Increasing Use in Science. The nature and features of coding errors make their existence a specific problem, which requires the best efforts to detect and correct them. Malfunctional objects are not always a problem. When an error manifests itself overtly, it can be fixed. For example, if a pen dries out, or its ink canal becomes congested, the effects are immediate: one can no longer write, and the pen is changed. Unfortunately, many errors in code are not of this kind. First, it is often the case that errors do not spoil the computational processes or the scientific results in ways that patently signal an underlying problem. Second, since scientists do not know in advance the precise results they should get, they are often in no position to know that something is

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

wrong. Third, even errors that have patent effects can be hard to identify. Finally, various errors create trouble only in particular circumstances, and may long go unnoticed until they have devastating effects. Overall, scientific codes are objects, the correctness of which can hardly be assessed *prima facie* by those who develop them. This can feed a misleading feeling of safety and, without appropriate safeguards, favor the development of sloppy science. Naturally, scientists keep developing various methods at the level of code or mathematics to detect code errors systematically. For example, backward error analysis, by estimating conditioning numbers and residuals,[5] allows scientists to detect that the numerical solutions of equations are deviating from the exact solution "even though the solution to the specified problem is *unknown*" (Fillion and Moir 2018, 747, our emphasis). Be this as it may, there is no general solution to the problem of detecting and fixing code errors, and assessing their impact, and this remains an ongoing worry. Scientific code is often of poor quality, and experimental studies have suggested that software is less accurate than generally believed (Hatton and Roberts 1994). Further, while good software is vital to research, funders, institutions, and practitioners often overlook its importance (Goble 2014, 4). Overall, scientific software errors can significantly impact published results (Hatton 1997; Merali 2010; Soergel 2015); thus, the existence of methods that can systematically and automatically check scientific code and increase its safety should be an attractive prospect for scholars interested in the reliability of computational inquiries.

[5] The condition number measures the stability of differential equations and the residual the extent to which the numerical solution is tangent to the vector field.

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

Formal verification methods are just such techniques, well-founded mathematically, automatically implemented on computers and designed to establish the correctness of programs or assist in the development of computer-based systems. They aim to guarantee that a computational system will function systematically as expected. They are generally associated with formal languages and formal semantics, used to describe the target properties of computational systems, and often come with powerful software tools to check whether these properties are instantiated by actual codes (Garavel and Graf 2013, 14; Butler and Muñoz 2016). Formal methods are no longer marginal: they now comprise a field of research investigated by a well-structured scientific community with scientific books, academic journals, societies, and regular international conferences. In 2011, more than one hundred formal method languages were already listed on the Formal Methods Wiki (Garavel and Graf 2013, 18). Overall, formal verification is a plural field of activity in which various complementary methods have been developed to guard against a range of computational errors.

These methods involve two main steps, viz. specification and verification (Clarke and Wing 1996). Specification consists in rigorously describing a computational system and its desired properties. One may indeed present specifications informally; however, in stringent applications, they are described in formal languages so that computer programs can take them as input for verification tasks. Then, formal tools can be used to analyze whether the computational system instantiates the target properties. Model-checking and theorem-proving are two such prominent methods (see, e.g., Karna et al. 2018). Model-checking consists in building a finite model of the computational system and checking

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

whether a target property holds in that model by performing an exhaustive state-space search. Theorem-proving consists in expressing the computational system as formulas in some mathematical axiomatized logic and then proving a statement corresponding to the target property within this framework. Other formal verification techniques include abstract interpretation or equivalence checking (Garavel and Graf 2013, 214; Campetelli 2010, 7).

Formal methods have now been under development for decades by mathematicians, logicians, and computer scientists. Because their potential pay-offs are high, debates about their scientific role and usefulness have regularly raged between their partisans and the pragmatists, who favored more direct and inquiry-relative ways of testing codes (Shapiro 1997). Until the 1980s or so, formal methods were an object of discussion mainly for computer scientists interested in analyzing to what extent computational results could be rigorously verified (see De Millo et al. 1979; Dijkstra 1978; Fetzer 1988 for influential contributions). The techniques themselves, at that early stage of development, could not provide exploitable tools. They had few applications, practitioners tended to consider them oddities with no nontrivial scientific applications, and they did not fulfill the hopes they raised. Thus, it was legitimate to consider that computational science, as a whole, was doomed to rely on informal methods to guarantee the reliability of its code (Shapiro 1997 and MacKenzie 2001).

However, there are good "reasons why now is a good time to revisit this hypothesis" (Fisher et al. 2017, 4), both within computer science and computational science. First, technological progress in computers has made a crucial difference, since faster processors

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

and computers allow scientists to adopt formal methods and apply them to huge search spaces. Secondly, there have been decisive improvements in automation, such as the development of SAT solvers to deal with problems of Boolean satisfiability.[6] Thirdly, formal methods have become more accessible, with formal-methods techniques, documentation, and licenses now publicly available. Fourthly, the growing complexity of computational code is setting the bar higher in terms of the techniques needed to guard against errors (ibid., 5). Finally, significant advantages have been reported to derive from their actual use in the world of industry: they improve the quality of products, reduce the time to market by enabling earlier detection of mistakes, and seem to reduce costs (Woodcock et al., 2009, Garavel and Graf 2013, 37). Overall, it is no surprise that major actors whose activity hinges on software reliability are increasingly using these methods: for example, the Amazon company adopted them in 2011 to help solve difficult design problems, and otherwise undetectable bugs were thereby resolved (Newcombe et al., 2015).

Importantly, the use of formal methods is also expanding and they are no longer mere oddities in the sciences, as highlighted by the prominent examples below. Airplanes represent highly critical systems involving many computers and automatic systems, where the absence of runtime errors is crucial. Unsurprisingly, these methods are used in

[6] SAT is a central decision problem dealing with the possibility of satisfying particular Boolean formulas. Because many problems reduce to SAT, advances concerning SAT can represent decisive breakthroughs.

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

avionics. For example, the Airbus company uses a formal method called Astrée to check the flight-control software of airplanes. Astrée is a static analyzer based on abstract interpretation. It allows Airbus to prove the absence of any runtime error in several safety-critical programs, e.g., in the primary flight control software of the Airbus A340 fly-by-wire system, a program of 132,000 lines of code (Garavel and Graf 2013, 31). Aerospace, more generally, is a field in which formal methods have achieved prominence. For instance, the French National Aerospace Research Centre "has chosen formal methods for the verification of critical aerospace software" because simulation and testing are "not exhaustive, and still very labor-intensive and costly", whereas formal verification techniques "have the advantage of being automated and exhaustive" (Wiels et al. 2012, 2). Formal verification techniques are now officially integrated into certification standards for avionics software, namely in the last version of the norm DO-178C that deals with "software considerations in airborne systems and equipment certification". The rationale for this replacement of testing techniques by formal verification methods is multi-faceted (Moy et al. 2013). For example, formal methods are well-suited for the assessment of software robustness (i.e., whether software can keep functioning under abnormal conditions). Time gains of "roughly a person-month per flight software release" (ibid., 55) are also achievable. Overall, the adoption of formal verification techniques "isn't simply possible but also practical and cost-effective, especially when backed by automated tools" (ibid., 55). The revision of the certification standard reflects a new awareness that "formal techniques can replace, and not simply augment, testing" (ibid., 56).

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

Large-scale computer simulations in hydrodynamics are another major case of the use of formal methods. For example, the ADvanced CIRCulation model (ADCIRC) is used by the U.S. Army Corps of Engineers as an ocean circulation model to study coastal flooding from tropical storms (see Jebeile and Ardourel (2019) for a philosophical presentation). For safety reasons, the reliability of these computer simulations is of great importance, so formal methods are employed to check crucial properties of the discretized mesh representing the ocean, which comprises 1,224,714 full finite elements. The formal method, called Alloy, consists of a declarative specification language with an automatic analysis performed by a SAT solver. Static and dynamic properties of the mesh are thereby checked, including topological relations and connectivity properties. Jebeile and Ardourel highlight several achievements of these methods, such as the verification of the discrete mesh, the absence of 'cut points' (i.e., regions where the width of the mesh is zero), or the dynamic evolution of triangles (viz., whether they change suitably from 'dry' to 'wet').

Finally, let us mention the use of formal methods to check the Clinical Neutron Therapy System, a complex neutron radiotherapy installation at the University of Washington Medical Center. It is presented as "a large-scale case study in applying modern verification techniques to check the safety of a radiotherapy system in current clinical use" (Pernsteiner et al. 2016, 24). The Alloy Analyzer was notably used to formally check different components and their interactions, and several safety-critical flaws were detected, demonstrating that formal methods "can provide significant practical benefits" by focusing "on deep properties of system components" (ibid., 16).

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

Are the above examples of the use of formal methods for code verification exceptions in empirical science or do they correspond to a larger trend? A recent empirical study offers insights into this issue (Padilla et al. 2018). The survey involved model builders from academia, private industry, and government, with various educational backgrounds. Several techniques of verification were reviewed, ranging from informal to formal methods. It turns out that simulation verification is mainly "a trial and error activity" but that formal methods come in second with 35.8% of respondents opting for the former, 21.2% for the latter, and 19.9% for visualization methods (ibid., 6). Moreover, this study reveals great disparities across domains. Formal methods are applied to systems dynamics models (30,5% of respondents), but hardly used for agent-based models (13.2% of respondents). Modelers in the defense (43.5%), healthcare (41.6%), and business industries "are more likely to formally verify their models", whereas modelers in science and engineering usually use systematic trial and error (ibid., 7).

Overall, these empirical findings highlight that the use of formal methods of verification is globally expanding in computational science, though very heterogeneously so. This is compatible with two scenarios. First, one might consider that the subfields in which they are already being used comprise a scientific avant-garde with respect to good computational practices, presaging the massive adoption of these methods across computational science. In this scenario, we are currently in a transient situation marked by the scattered use of these methods by pioneers, accompanied by steady development. Alternatively, it may be the case that these methods, though developing in science, cannot become a common standard, and are doomed to remain restricted to particular (albeit

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

sometimes large) environments that are well-suited for their use. We maintain, below, that the second scenario obtains.

4. Obstacles to the Development of the Use of Formal Methods. Formal methods of verification have significant advantages; why, then, if they are so beneficial, would not all fields adopt them? We argue in this section that these methods also come with various major inconveniences. We characterize the nature of these inconveniences, highlight their diversity, ranging from logical to social or organizational constraints, and emphasize that these constraints are often well-entrenched and unlikely to disappear in the future with the progress of science.[7]

To start with, the benefits of formal methods, though substantial, also have intrinsic limitations. Firstly, formal methods can warrant only that codes correctly fulfill existing formal specifications. Thus, they are conclusive only if these specifications adequately catch the target problems. Yet, whether the latter is the case cannot be decided by formal

[7] From a methodological perspective, the diverse nature of these constraints implies that their description and analysis must draw on different sources in addition to research papers about formal methods, typically grey literature (viz. technical reports where techniques and methods are discussed) or scholarly studies which analyze how science and scientific communities work. This orientation is consistent with the recent naturalistic and practical turns in philosophy of science. We thank an anonymous reviewer for highlighting this point.

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

means, since this requires determining whether a formal object adequately corresponds to an informal goal. Further, when writing specifications, scientists cannot be certain that all relevant aspects and sources of troubles have been taken into account. Accordingly, verification proofs of correctness are always relative to particular specifications, and it is in the specifications that the problems often lie. Practitioners in this field acknowledge these limitations: "[formal methods] can greatly increase our understanding of a system by revealing inconsistencies, ambiguities, and incompleteness that might otherwise go undetected", but their use "does not *a priori* guarantee correctness" (Clarke and Wing 1996, 626). Moreover, even when errors are detected, it may be hard for practitioners to obtain valuable systematic feedback concerning how to handle errors successfully (Heitmeyer 2004, 15). In other words, formal methods negatively certify the absence of *some* causes of failure but do not provide general positive warrants about the reliability of computational systems.

Secondly, many verification activities involve tackling undecidable problems. Thus, tractable solutions for particular verification problems do not always exist, and existing algorithms can solve only restricted verification problems (Garavel and Graf 2013, 16). Accordingly, the benefits of formal verification are not always accessible.

Thirdly, formal methods of verification are not general-purpose tools. Many different properties need to be verified, whereas verification methods are targeted at specific types of properties: for example, some can be verified with model-checking or abstract interpretation, others by theorem-proving. Further, the properties to be verified potentially evolve along with the flow of design (Garavel and Graf 2013, 214). Finally, formal

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

verifications techniques are usually relative to particular languages and have mostly been developed for specific languages: for example, Lint is a static analysis tool used to detect errors in C and Unix programs, and CodeSonar is a tool to check for template errors in C, C++, and Java code.[8] Conversely, most coding languages are not designed for verification, and language chosen constrains the available methods and the set of accessible verification proofs. Overall, there is both a wealth of verification tasks and of methods with different and limited potentials. The prospects of developing all-purpose and easy-to-use verification methods are weak, and the verification of code seems to be doomed to remain a patchwork activity.

Finally, benefiting from formal methods may imply forgoing other epistemic benefits. As Wayne (2019) claims: "What's good for coding is bad for proving! Formal verifiers have a dilemma: the more expressive the language, the harder it is to prove anything in it. But the less expressive the language, the harder it is to write anything in it" (2019, sect. 'Proofs are hard'). Thus, there is a trade-off between the epistemic affordances of coding languages with respects to other epistemic virtues, and their potential for formal verification. This point raises particular challenges in the context of the development of accessible coding practices and user-friendly languages, since these may not be suitable for formal verification. Overall, a situation in which practitioners can easily enjoy all the

[8] Despite recent works in this direction (see, e.g., Liu et al. 2020), formal verification techniques seem to be less developed for widely used programming languages like Matlab and Python.

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

advantages of formal methods for all their verification needs does not seem a close prospect.

The other main worry is that using these methods safely remains very costly in terms of time, skills, and knowledge. Firstly, both the application of partial and complete verification of codes and programs, where this is indeed possible, remains highly time-consuming, which explains why they are often restricted to critical scientific domains. Clearly, users might be satisfied if, e.g., only 90% of their code is formally verified, given that formal verification comes in degrees, and the marginal costs to further verify codes (say, to verify 99% instead of 95%) tend to be increasing. However, partial verification does not escape issues pertaining to languages mentioned above, which explains why it has thus far focused mainly on a few high-priority languages, such as C and Java (Wayne 2019, sect. 'Partial verification'). Thus, being more flexible about reliability thresholds alleviates the costs but does not make formal methods a widely accessible tool.

Secondly, each formal method exacts its specific heavy toll in terms of learning costs. Indeed, it requires resorting to particular formal tools, which have their technicalities and are not part of practitioners' culture in empirical science. Further, because there are no all-purpose formal methods, practitioners cannot learn one such technique and live happily for the rest of their computational lives; instead, they are called upon to develop knowledge of the various existing methods so as to be in a position to choose and apply them contextually. In brief, using formal methods significantly extends the epistemic burden of those trying to use them to study natural systems through computational inquiries. Can we expect most scientists to make these efforts? Probably not. Remember that, in general,

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

scientific code is already of low quality (Wilson et al. 2014, 1). The lack of coding training, the pressure to publish quickly, and the modalities of project funding already provide strong incentives to neglect the writing of good code. Thus, the development of good *and* formally verified codes is even more unlikely.

A third inconvenience under this heading is that formal methods require extreme clarity in advance about the target properties that are to be tested, as well as clear oversight of the whole scientific process to avoid working for nothing. But "scientific software is […] often explorative: the purpose of the software is usually to help to understand a new problem, implying that up-front specifications of software requirements is difficult or impossible" (Hannay et al. 2009, 1). Further, because of their lack of scalability, investing in formal methods remains risky for researchers in contexts of uncertainty. In brief, using formal methods adds strong additional constraints on scientific inquiry and does not square well with how research is often conducted.

And finally, significant contingent factors may deter individual scientists from using these methods. It is easier and safer to use complicated techniques when one has the backing of one's community, which valorizes their use, encourages researchers to apply them, provides shared facilities to make their adoption less costly, and institutionalizes their learning by apprentices. As for various techniques, the costs and benefits of these methods depend on the environment in which they are used and whether common resources are already devoted to fulfilling the corresponding standard. Unsurprisingly, models that are used jointly by connected teams are more often formally verified (Padilla et al. 2018, 498-499). In a nutshell, there is no free lunch when using these methods, which

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

remain costly, both for individuals and for the communities that provide a sound environment and resources to facilitate their use.

Let us take stock. We have listed and characterized several kinds of significant obstacles to the use of formal methods. While formal verification methods are high-profile techniques which address the crucial goal of verifying scientific code, they are also, as shown above, cumbersome, epistemically demanding, high-maintenance, and poorly scalable. Many practitioners, therefore, are likely to consider these methods entirely unattainable. In this context, it is no surprise that major actors of the Verification and Validation community tend to discourage engineers from using them. Oberkampf and Roy, for example, briefly note that they "do not recommend formal methods for scientific computing software" due to the "effort and expense required, as well as their poor scalability" (2010, 159). The additional facts that the listed obstacles are well-entrenched, unlikely to disappear, and are operative at both the individual and the community level vindicates our additional claim that these methods are unlikely to become widespread across computational communities. Accordingly, the elaboration of error-proof codes seems to be doomed, in large part, to remain an activity that scientists will continue to pursue armed with the lore of non-formal contextual techniques developed over the decades to verify code.

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

5. Profiling the Scientific Uses of Formal Methods.

Yet even if there are strong reasons to discourage the *general* use of formal verification methods and to doubt that they will become part of the standard toolkit of computational practitioners, it does not follow that they should be dismissed in all scientific contexts or that their inconveniences are always insuperable. We now combine the insights drawn from the above cases and analyses, and sketch the types of contexts in which these methods may be adapted, typically in terms of the inquiries pursued, amenability to standardization, available resources, and organization of the corresponding communities.

First, the standardization of inquiries can remove much of the inconvenience of the resort to formal methods. Many investigations are not exploratory nor carried out by researchers working alone. Indeed, a progression towards calibrated inquiries is rather common in the development of a science. In periods of "normal science", investigated questions are clearly circumscribed and large communities tackle them (Kuhn 1962; De Langhe 2014). In such contexts, scientists usually know in advance the target properties to be verified and can plan the time-consuming development of formally verifiable code early in the pipeline. Further, the use of formal methods is less adventurous in the context of communities that already recognize these methods as part of shared consensus practices, which include typical methods and standard norms (Kitcher 1983). Finally, economies of scale that lead to favorable cost/benefit ratios are possible for standardized investigations, in which codes can be used repeatedly.

Secondly, because formal methods increase the epistemic burden of computational inquiries, their adoption is easier in fields with abundant financial and human resources where scientific labor can be suitably divided, and expert collaborators can take charge of

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

developing and formally verifying code. Similarly, within well-organized communities, the learning of these methods can be anticipated, institutionally pooled, and facilitated by dedicated facilities.

Thirdly, these methods have to be worth the effort, given their users' profiles and the inquiries they pursue. The cost/benefit ratio of formal methods can vary significantly with respect to their target applications or to how much scientific success or failure is positively or negatively valued. For fixed costs, they are more attractive when a lot is at stake and benefits are high. This may obtain in several types of cases, for example, when applied "to the hardest and most difficult problems" (Rushby 1997, 18). Typically, using these methods may be worthwhile for high-profile mathematics, e.g., to prove famous conjectures in mathematics which have partly computational proofs. More generally, computationally costly inquiries are another suitable niche for formal methods since the risk of "not conducting verification formally and accepting models as verified increases with the complexity of the models" (Padilla et al. 2018, 500). It may be worthwhile to verify the code by stringent methods when a second chance is not an option, and the software must work immediately (e.g., when researchers obtain a valuable competitive slot in a computational center and several runs are not possible). Unsurprisingly, the profiles of the prominent examples cited above fall within some of these categories.

Nevertheless, all the above conditions are far from being met across science. Various fields are not standardized with respects to the forms of the problems that are investigated nor the methods deployed to tackle them. Much scientific research is practiced in an exploratory way that does not square well with formal methods. Funding and human

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

resources are not equally distributed across fields. Many scientific inquiries are not associated with high computational costs, major scientific gains, or significant financial or human losses in cases of errors. Further, trying to bypass existing difficulties and insisting on using formal verification methods can be counterproductive because their *felicitous* use requires deploying a wealth of resources, sidestepping various pitfalls, and is challenging. In brief, not only can the attempts to apply these methods in unfavorable contexts be even more costly, but they may also be fruitless and lead to unreliable results by non-expert practitioners using too-complicated methods. This may be a waste of scientific resources, especially if other simpler-to-tame methods can increase our confidence in computational codes up to an appropriate level. (Here, remember again that non-formal methods for code verifications, though imperfect, have been refined over the years by practitioners and already have significant merits (see, again, Shapiro 1997).)

Importantly, the above features, which make differences concerning the unequal suitability of formal methods, are somewhat structural. Science involves activities that are often exploratory concerning their goals, problems to be tackled, or methods. And the various benefits and risks related to scientific inquiries and their applications cannot be homogeneous since the scientific or social value attached to scientific findings can vary significantly. Furthermore, fundings and human resources can hardly be evenly distributed across science. In brief, then i) well-entrenched intrinsic features are an obstacle to the use of formal methods (see section 4); and ii) the features that may encourage scientists to use them still are not present across science, and this lack of homogeneity is also well-entrenched too. Thus, even if formal methods do keep developing and become somewhat

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

more accessible, they are still unlikely to become mainstream, and it seems to be a well-entrenched fact that verification across science will continue to be carried out in deeply heterogeneous ways that are both formal and non-formal.

Overall, our argument mainly relies on the inherent difficulty of singling out and eliminating error in scientific code, on an analysis of the intrinsic advantages and limitations of formal methods, on the existence of other valuable, though imperfect methods for code verification, and on some deeply-rooted features of scientific research and communities. Given the lack of knowledge about the merits of future verification methods, it would be risky to make predictions about the future development of formal verification methods, since the attractiveness of scientific methods depends on their comparative merits. The present appeal of formal methods may even decline, at least in specific fields, if other user-friendly and powerful methods develop. For example, *a posteriori* methods based on residual analysis and backward error (see supra) may become more attractive whenever differential equations are involved. Indeed, these methods can be implemented automatically and used "on the fly" as the problem is being solved or after the computation is completed, which makes them flexible and appealing tools. Be this as it may, absent significant changes concerning central features of formal methods or the advent of novel, versatile, and user-friendly verification methods, the above-noted heterogeneity concerning how code is verified is unlikely to change.

Let us note, finally, that our arguments are primarily rooted in an analysis of the *practical* advantages and inconveniences of formal methods. As such, they do not involve commitments concerning the nature of mathematical proofs and whether formal methods

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

qualify as such (see, e.g., Dijkstra 1978; De Millo et al. 1979; Mac Kenzie 2001, 316). However, they highlight that practical constraints bear on what epistemic agents actually do, can do, or should do. As such, they belong to a tradition emphasizing that overly idealized descriptions of agents and their capacities can be inadequate for epistemological inquiries (see, e.g., Simon 1969; Grandy 1994; Humphreys 2004).

6. From scientific heterogeneity to epistemological heterogeneity

We now emphasize that this situation of scientific heterogeneity concerning how code is verified also qualifies as one of epistemological heterogeneity. Epistemology is a critical domain of evaluation that analyzes how epistemic processes work and which factors make differences to the quantity and quality of their outputs (see, e.g., Sosa 2007, 73). A key question in this domain is whether, and if so when, overtly dissimilar practices work in the same underlying epistemic ways. Note, in this respect, that an account implying that different scientific practices always have different epistemologies would be trivial and non-informative (non-triviality condition).

From this perspective, a general typological strategy is to conceptualize scientific practices so that individual activities can be classified within larger coherent clusters highlighting features that make epistemological differences and pointing out homogeneous scientific cultures. For example, Kitcher describes scientific (including mathematical) practices as multi-dimensional entities involving i) scientific languages, ii) accepted statements, iii) accepted schemas of reasoning, iv) questions selected as important, and v)

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

methodological views about the canons of good practices for assessing reliability and achieving success, typically standards for proofs in mathematics (Kitcher 1993, 74; Kitcher 1983, 63).

This strategy provides a means to highlight when different practices work in epistemologically similar ways and in which cases different scientific practices are scientifically and epistemologically distinct – namely when they involve specific languages, questions, or canons of good practices. This is the case with formal and non-formal code verification methods, which can be described as epistemologically distinct practices. Consistent with this perspective, MacKenzie characterizes the former as a specific "culture of proving" (2001, 306). Further, since verification methods aim at establishing the reliability of computational code, they belong to the fifth normative dimension described above. As such, differences in verification practices and standards are pivotal features which can be used to classify computational activities within epistemologically similar or different clusters, depending on how they verify code. In particular, practices within a common field (e.g., in computational fluid dynamics) can be epistemologically heterogeneous if they are verified differently.

Such general typological strategies reveal the global epistemological coherence or dissemblance of scientific practices and cultures. However, finer-grained and more piecemeal analyses can also prove useful, in particular to point out similar epistemological features that may be present across different practices. For example, genuinely *distinct* scientific practices from different domains like biology or physics (in the above Kitcherian sense) may be epistemologically *homogeneous* concerning their epistemic sources (e.g.,

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

experiments, testimony, or theories), norms of good research practices (e.g., different techniques for measuring a mass in physics may fulfill common requirements concerning the nature of good measurements), or the problems that they face (e.g., experimental and computational activities raise identical issues concerning data or instruments). This shows the need for a joint analysis of computational practices that belong to different fields but that are (partly) epistemologically homogeneous, in particular if they comply with identical verification standards. Conversely, one must be aware that practices may raise identical problems but call for distinct analyses if they tackle them in significantly different ways. In the present case, scientific code is used across science, is involved in different scientific practices, and raises closely related questions. Nevertheless, strong heterogeneities remain concerning the procedures and norms applied to solve coding issues successfully, here concerning how to verify code.

Overall, all these analyses show how the non-triviality condition is met. However, let us keep in mind that epistemological heterogeneities can be weaker or stronger. Arguably, the differences in code verification described above qualify as strong, even if disagreements may remain concerning how exactly to conceptualize them and where to draw the lines. Indeed, differences regarding verification standards imply differences concerning the norms of good research practices. These globally constrain activities and have far-reaching effects concerning issues such as which skills the inquiry requires, who writes the code, how to organize teams and divide labor, how computational models are elaborated, and which types of errors may remain. Further, as pointed out above, the existence of different verification standards may be rooted in practical reasons and need not imply major

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

disagreements concerning basic notions like proofs or evidence. Finally, practices may be epistemologically heterogeneous with respect to some aspects and homogeneous and part of common cultures with respect to others. Thus, the existence of strong epistemological heterogeneities within computational science need not imply any global epistemological disunity between the diverse parts of computational science that would call for utterly different epistemologies.

7. Methodological Consequences. We now draw methodological morals concerning how to pursue the epistemology of computational practices. There is a growing awareness that computational methods raise similar problems across fields. Examples include the writing of safe code, its verification, the production of reproducible computational research, the management of data, the opacity of computational methods, the use of partly automated procedures, the use of good pseudo-random numbers, or the visualization of results. Should all these issues receive general treatments that apply to all computational practices?

As argued above, the use of formal verification methods seems appropriate only in specific situations. This provides a clear example of scientific and epistemological heterogeneity, and of the need for contextual analyses that reflect this variety. This also illustrates how problems can be common across computational science yet different concerning how they are, can, or should be addressed by practitioners. Finally, this suggests that, by default, one should guard against assuming that the above questions call

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

for unique answers across contexts. Clearly, it is legitimate to try to offer epistemological analyses that are not overly specific nor blind; however, unduly general claims with a too vague and unwarranted scope run the risk of being refuted by careful analyses that are sensitive to the variety of ways in which scientists pursue computational inquiries. They can even trigger spurious counterproductive controversies in which examples rooted in different fields are exploited by the contending parties to back up too-general claims. Indeed, it is not hard to imagine how a 'dialogue of the deaf' concerning the general possibility of using formal methods could be fed by the employment of heterogeneous case studies.

In brief, even when they are addressing identical problems, computational scientists can take heterogeneous pathways across scientific contexts. Our epistemological analyses of computational science ought to reflect such existing heterogeneities and provide contextual insights instead of being squeezed into a common straight-jacket. However, the case for heterogeneity should not be taken for granted either, and emphasizing that different contexts call for different epistemological analyses itself threatens to become a lazy stance. As already noted, identical principles, norms, or mechanisms can often be at work beyond the overt diversity of contexts and practices.

Overall, the extent to which the challenge of code verification is a typical issue and the epistemology of computational science is indeed heterogeneous remains an open question. For example, do claims about justification holism in simulations or the difficulty of disentangling verification and validation in computational science (see section 2) call for

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

more contextual analyses (formal verification is a way to disentangle things in part, after all)? Similarly, can the insights by Lenhard (2018) about the tendency of the modularity of computational codes to erode be generalized? Or, are they only valid when codes are repeatedly reused, and conditions related to the "economics of the software" within commercial science are met? (See Foote and Yoder 1999 for an informed discussion.)

Unfortunately, it is difficult to determine in advance which epistemological questions call for context-specific answers and which may be given general answers. By default, a way out of the deadlock is to require that the domain of validity of the arguments and evidence that support epistemological claims be clearly highlighted, so as to make their scope noncontroversial.

Let us wrap up. It is now a matter of consensus that general issues in the philosophy of science may call for distinct answers as regards physics, biology, social science, etc. The same potentially holds true within the philosophy of computational science, where idealized and acontextual epistemological pictures that ignore existing specificities may require more than simple corrections. Concretely, epistemologists of computational science should try to draw a fine-grained map of the features that make genuine epistemological differences concerning how computational inquiries function across contexts. Thereby, they can hope to draw a line between issues that belong to the general epistemology of computational science, and those that call for practice-related or context-sensitive treatments.

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

8. Conclusion. For decades, the formal verification of computational codes was little more than a promissory note, and practitioners had no choice but to resort to non-formal methods to test their codes. Now things have changed, and the formal verification of many aspects of code can be carried out in actual scientific practice. However, these methods come with major and deeply rooted inconveniences, and seem best suited for specific types of contexts, sometimes broad, sometimes less so, but not well-adapted for the large variety of situations in which computational science is developing. Typically, their adoption is easier for research fields that are standardized, well-structured, well-funded, and engage in repeat investigations of similar problems. Conversely, it seems more hazardous for small groups and communities, for less well-funded fields, or for explorative science in which inquiries may take unexpected turns, research topics may be mutable, and problems are not standardized.

Overall, while code verification is a global problem shared by all practitioners in computational science, the solution to this problem goes through profoundly different epistemic "pathways" (Goldman 2002) across contexts. This illustrates how computational science is partly developing into cultures that, at their core, are both scientifically and epistemologically heterogeneous. Thus, it would be risky to assume by default that computational science has a common epistemology, even when it comes to transversal questions like code verification. It remains to investigate how deep and pervasive this heterogeneity may turn out to be in a context where the development, diversification, and popularization of computational tools is constantly increasing the variety of their uses.

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

9. References

Beisbart, Claus, and Nicole J. Saam. 2019. ed. *Computer Simulation Validation: Fundamental Concepts, Methodological Frameworks, and Philosophical Perspectives*. 1st ed. New York, NY: Springer.

Beisbart, Claus. 2019. "Should Validation and Verification Be Separated Strictly?" in Beisbart and Saam, ed., 2019, 1005-28.

Butler, Richard, and César Muñoz. 2016. "What Is Formal Methods?" *NASA Langley Formal Methods Research Program*. last update. <https://shemesh.larc.nasa.gov/fm/fm-what.html>.

Campetelli, Alarico. 2010. *Analysis Techniques: State of the Art in Industry and Research.* TU München. <https://mediatum.ub.tum.de/doc/1094378/file.pdf>

Clarke, Edmund M., and Jeannette M. Wing. 1996. "Formal Methods: State of the Art and Future Directions." *ACM Computing Surveys* 28: 626–643.

Creel, Kathleen A. 2020. Transparency in Complex Computational Systems, *Philosophy of Science*, 87(4). <https://doi.org/10.1086/709729>

De Langhe, Rogier. 2014. "A Unified Model of the Division of Cognitive Labor." *Philosophy of Science* 81, no. 3: 444–59.

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

De Millo, Richard A., Richard J. Lipton, and Alan J. Perlis. 1979. “Social Processes and Proofs of Theorems and Programs.” *Commun. ACM* 22, no. 5: 271–80.

Dijkstra, Edsger W. 1978. “On a Political Pamphlet from the Middle Ages.” *ACM SIGSOFT Software Engineering Notes* 3, no. 2: 14–16.

Fetzer, James H. 1988. “Program Verification: The Very Idea.” *Commun. ACM* 31, no. 9: 1048–63.

Fillion, Nicolas, and Robert Corless. 2014. “On the epistemological analysis of modeling and computational error in the mathematical sciences.” *Synthese*, 191, 1451-1467.

Fillion, Nicolas. 2017. “The Vindication of Computer Simulations.” in Lenhard, J. and Carrier, M. eds. Mathematics as a tool, *Boston Studies in the Philosophy of Science*: 137-156.

Fillion, Nicolas, and Robert. H. C Moir. 2018. “Explanation and abstraction from a backward-error analytic perspective.” *European Journal for Philosophy of Science*, 8 , 735-759.

Fisher, Kathleen, John Launchbury, and Raymond Richards. 2017. “The HACMS Program: Using Formal Methods to Eliminate Exploitable Bugs.” *Philosophical Transactions. Series A, Mathematical, Physical, and Engineering Sciences* 375 (2104).

Frigg, Roman, and Julian Reiss. 2009. “The Philosophy of Simulation: Hot New Issues or Same Old Stew?” *Synthese* 169 (3): 593–613.

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

Foote, Brian and Joseph Yoder. 1999. “Big Ball of Mud.” In *Pattern Languages of Program, Design 4*, ed. Neil Harrison, Brian Foote and Hans Rohnert, 1-41. Boston: Addison-Wesley.

Fresco, Nir, and Guiseppe Primerio, 2013, Miscomputation, *Philosophy and Technology*, 26/3: 253-272.

Garavel, Hubert, and Susanne Graf. 2013. “Formal Methods for Safe and Secure Computers Systems: BSI Study 875.” Bonn: Federal Office for Information Security.

Goble, Carole. 2014. “Better Software, Better Research”, *IEEE Internet Computing* 18, nº 5.

Goldman, Alvin. 2002. *Pathways to Knowledge: private and public*, Oxford University Press.

Grandy, Richard, 1994, “Epistemology Naturalized and “Epistemology Naturalized””, *Midwest Studies in Philosophy*, XIX: 341–349.

Hannay, Jo E., Carolyn MacLeod, Janice Singer, Hans P. Langtangen, Dietmar Pfahl, and Greg Wilson. 2009. “How Do Scientists Develop and Use Scientific Software?” In *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, 1–8.

Hatton, Leslie, and Andy Roberts. 1994. “How accurate is scientific software?” *IEEE Transactions on Software Engineering* 20, (10): 78597.

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

Hatton, Leslie. 1997. “The T-Experiments: Errors in Scientific Software”. In *Quality of Numerical Software*, 1231. IFIP Advances in Information and Communication Technology. Springer, Boston, MA.

Heitmeyer, Constance. 2004. “Managing complexity in software development with formally based tools.” *Electronic Notes in Theoretical Computer Science*, 108: 11–19.

Humphreys, Paul. 2004. *Extending ourselves: Computational science, empiricism, and scientific method*. New-York: Oxford University Press.

Humphreys, Paul. 2009. “The philosophical novelty of computer simulation methods.” Synthese, 169(3):615-626.

Jebeile, Julie, and Vincent Ardourel. 2019. “Verification and Validation of Simulations Against Holism.” *Minds and Machines* 29 (1): 149–168.

Karna Anil K., Yuting Chen, Haibo Yu, Hao Zhong, and Jianjun Zhao. 2018. ‘‘The role of model checking in software engineering’’, *Frontiers of Computer Science*, 12 (4): 642–668.

Kitcher, Philip. 1983. *The Nature of mathematical knowledge*. Oxford University Press.

Kitcher, Philip. 1993. *The Advancement of Science: Science without Legend, Objectivity without Illusions*. New York: Oxford University Press.

Kuhn, Thomas Samuel. 1962. *The Structure of Scientific Revolutions*. Chicago: University of Chicago Press.

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

Lenhard, Johannes. 2018. “Holism, or the Erosion of Modularity - a Methodological Challenge for Validation”, *Philosophy of Science*, 85, 5, 832-844.

Lenhard, Johannes. 2019. *Calculated Surprises: A philosophy of Computer Simulation*. Oxford Studies in Philosophy of Science.

Lenhard, Johannes and Küster, Uwe. 2019. “Reproducibility and the Concept of Numerical Solution.” *Minds & Machines* 29, 19–36.

Lions, Jacques-Louis. 1996. “Ariane 5 Flight 501 Failure.” *Ariane 501 Inquiry Board Report*, URL = <https://esamultimedia.esa.int/docs/esa-x-1819eng.pdf>.

Liu, Ailun, H. Zhu, M. Popovic, S. Xiang, and L. Zhang. 2020. “Formal analysis and verification of the PSTM architecture using CSP.” *Journal of Systems and Software*, 165, 110559.

MacKenzie, Donald. 2001. *Mechanizing Proof: Computing, Risk, and Trust*. Cambridge, Mass.: MIT Press.

Merali, Zeeya. 2010. “Computational science: ...Error”, *Nature* 467, n^{o} 7317.

Morrison, Margaret. 2014. “Values and Uncertainty in Simulation Models.” *Erkenntnis* 79 (S5): 939–959.

Morrison, Margaret. 2015. *Reconstructing reality: models, mathematics, and simulations*. Oxford University Press.

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

Moy, Y., Ledinot, E., Delseny, H., Wiels, V., & Monate, B. 2013. Testing or Formal Verification: DO-178C Alternatives and Industrial Experience. *IEEE Software*, 30(3), 50–57.

Newcombe, Chris, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. 2015. "How Amazon Web Services Uses Formal Methods." *Communications of the ACM* 58 (4): 66–73.

Oberkampf, William L., and Christopher J. Roy. 2010. *Verification and Validation in Scientific Computing*. Cambridge: Cambridge University Press.

Padilla, Jose J, Saikou Y Diallo, Christopher J Lynch, and Ross Gore. 2018. "Observations on the Practice and Profession of Modeling and Simulation: A Survey Approach." *Simulation* 94 (6): 493–506.

Pernsteiner, Stuart et al. 2016. "Investigating Safety of a Radiotherapy Machine Using System Models with Pluggable Checkers." In *Computer Aided Verification*, edited by Swarat Chaudhuri and Azadeh Farzan, 23–41, Springer.

Rider, William J. 2019. ''The Foundations of Verification in Modeling and Simulation'', in Beisbart, Claus and Saam, Nicole J. ed. 2019, *Computer Simulation Validation*, Springer.

Rushby, John. 1997. "Formal Methods and Their Role in the Certification of Critical Systems." In *Safety and Reliability of Software Based Systems*, ed. Roger Shaw, 1–42. Springer London.

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press

Rushby, John 2007. ‘‘Automated Formal Methods Enter the Mainstream’’, *Journal of Universal Computer Science*, vol. 13(5): 650–660.

Shapiro, Stuart. 1997. “Splitting the Difference: The Historical Necessity of Synthesis in Software Engineering.” *IEEE Ann. Hist. Comput.* 19 (1): 20–54.

Simon, Herbert A. 1969. *The Sciences of the Artificial*. Cambridge, MA: The MIT Press.

Soergel, David A. W. 2015. “Rampant Software Errors May Undermine Scientific Results.” *F1000Research* 3: 303.

Sosa, Ernest, 2007. *A virtue epistemology. volume I, apt belief and reflective knowledge*: Oxford University Press.

Wayne, Hillel. 2019. “Why don't people use formal methods?”, personal blog: <https://www.hillelwayne.com/post/why-dont-people-use-formal-methods/>

Wiels, Virginie, Robert Delmas, David Doose, Pierre-Loïc, Garoche, J. Cazin, and Guy Durrieu. 2012. “Formal Verification of Critical Aerospace Software.” *AerospaceLab*, no. 4 (May): 1–8.

Wilson, Greg, D. A. Aruliah, C. Titus Brown, Neil P. Chue Hong, Matt Davis, Richard T. Guy, Steven H. D. Haddock, Kathryn D. Huff, Ian M. Mitchell, Mark D. Plumbley, Ben Waugh, Ethan P. White, Paul Wilson. 2014. “Best Practices for Scientific Computing.” *PLOS Biology* 12 (1).

Winsberg, Eric. 2010. *Science in the age of computer simulation*. Chicago: University of Chicago Press.

Woodcock J., Larsen P. G., Bicarregui J. and Fitzgerald. 2009. ‘‘Formal Methods: Practice and Experience’’, *ACM Computing Surveys*, 41(4), 19: 1–36.

https://doi.org/10.1017/psa.2022.78 Published online by Cambridge University Press