Formal proof and trust

Dale Miller
Inria-Saclay & LIX, Ècole Polytechnique, France
July 16, 2019

Short abstract

While formal proofs can be a source of trust, I will argue that the very notion of formal proof requires other sources of trust. In fact, formal proofs, as documents, are produced by machines and are checkable only by machines and, as a result, we must trust the many concomitant components associated with proof checking. Such components include, for example, printers and parsers of such structured documents, the compiler and run-time systems that supports a programming language, and some machine hardware. Thus, trusting a formal proof relies on the correct functioning of components for which we have ample evidence of their shortcomings vis-à-vis correctness. Since many approaches to formal proofs provide such structures with well-understood syntax and semantics, the need to trust particular computer systems is either ameliorated or entirely removed by realizing that proof checking is *reproducible*. Just as "*nullius in verba*" (take no one's word for it) is a slogan of the scientific method, I argue that this slogan makes serious demands on the design of proof checkers and of formal proofs.

Extended abstract

In mathematics, a careful proof provides trust in a theorem. With such trust, we take actions such as publishing a paper, starting a new research effort, or building a physical object that we expect to work in a specific fashion. The checking of the correctness of a proof is a central activity of any discipline that makes use of proof, particularly, mathematical proofs. Humans (e.g., reviewers) are often used as proof checkers and, in many ways, mathematical proofs allow for a kind of communication between humans.

Recent decades have witnessed a rise in the use of formal proofs for various purposes. One such use of formal proofs allows for large and complex mathematical proofs (of, say, the Feit-Thompson Theorem) to be checked with a level of detail not usually achievable by humans. Formal proofs have also been built for mathematical theorems for which human would find it nearly impossible to consider carefully all the necessary cases (for example, the proof of the Kepler conjecture and of the four color theorem). Finally, it seems that only formal proof will allow us to prove various properties of software and hardware systems.

For my purposes here, a formal proof is a document (a computer file) that contains enough information so that a relatively simple proof checker can construct a fully formal proof in a well-established style of proof, such as Frege/Hilbert proofs, sequent calculus proofs, natural deduction proofs, tableaux proofs, or resolution refutations. If such documents are structured so that all the information needed to build explicit proofs are present, then proof checkers can be simple and small programs. On the other hand, if such documents contain only some information about a proof, then a proof checker will need to be able to reconstruct the missing details: such checkers are more complex pieces of software. In either case, however, formal proofs are documents that are produced by machines (sometimes as the result of interactions with humans) and that are checked by machines. In general, we do not expect humans to read, understand, and check a formal proof.

It seems that we must, therefore, trust in the correctness of proof checkers and the computer systems on which they are implemented. Since our universal experience with computer systems is that they can be riddled with errors, we can easily doubt proof checkers and, consequently, doubt whether or not a given checked document is, in fact, a formal proof. This raises the familiar and ancient conundrum "Quis custodiet ipsos custodes?" (Who will guard the guards?). Fortunately, there is a modern approach to addressing this problem: make it possible for anyone and everyone to monitor and audit the guards (proof checkers, in our case). Thus, we invoke a well-known approach to trust which comes directly from the *scientific method*: in particular, proof checking needs to be *reproducible* in the sense that it is an activity that anyone should be able to undertake, now and in the future. Thus, while proof checking depends on technology it should not be dependent on any specific technology: 50 years from now when computer hardware and programming language technologies have radically changed, it should be possible to rebuild proof checkers on that newer technology so that a future skeptic can recheck proofs.

Of course, to communicate a proof to, say, a skeptic 50 years from now, some carefully defined standards for describing formulas and their proofs need to exist. Both logical inference and formal proof have been studied extensively during the past several decades for first-order and higher-order versions of classical and intuitionistic logics. For example, the notion of a theorem in first-order logic can be describe in multiple ways, including using model theory and using a variety of proof structures. Furthermore, there are numerous papers and implementations of the basic algorithms that underlie proof search and proof checking. It is easy to find notions of formalized reasoning for which there is nothing ad hoc and temporary: anyone 50 years from now will be able to understand exactly the same notion of theorem as we understand today. That foundation provides the basis for writing a clear, flexible, and permanent definition of the syntax and semantics of formulas and proofs.

Some communities within the computational logic field have already been establishing such standardize certificate formats. For example, researchers building systems that determine whether or not a propositional formula is satisfiable have designed various standardize proof certificate formats than their search programs can output. These formats have names such as DRUP [3] and DRAT [5] and anyone can build rather simple checkers that can check if a file in one of these formats actually describes a proof. Other researchers working in rewriting systems have designed the CPF proof certificate format [4] that can be used to check whether or not a given rewrite system is terminating or confluent.

Broad spectrum certificates are also under development. The Dedukti system [1] mixes functional programming style rewriting with dependently typed λ-calculus to provide a proof certificate for the various theorem provers working in higher-order intuitionistic logic. *Foundational proof certificates* [2] use logic programming techniques to check a range of proof systems in classical and intuitionistic logics by allowing for proof reconstruction during proof checking. In both of these cases, the underlying proof checking technology is based on well-defined and understood computational logic frameworks: our descendants will be able to build their own proof checkers and recheck any of our proofs written in these technology independent formats.

References

[1] A. Assaf, G. Burel, R. Cauderlier, D. Delahaye, G. Dowek, et al., Expressing theories in the $\lambda\pi$-calculus modulo theory and in the Dedukti system. In *TYPES: Types for Proofs and Programs*, Novi Sad, Serbia, 2016.

[2] Z. Chihani, D. Miller, and F. Renaud. A semantic framework for proof evidence. *J. of Automated Reasoning*, 59:287–330, 2017.

[3] M. Heule, W. A. Hunt Jr, and N. Wetzler. Expressing symmetry breaking in DRAT proofs. In A. P. Felty and A. Middeldorp, editors, *Automated Deduction - CADE-25*, LNCS 9195, 591–606. Springer, 2015.

[4] C. Sternagel and R. Thiemann. The certification problem format. In *Proceedings UITP 2014*, 61–72, October 29 2014.

[5] N. Wetzler, M. J. H. Heule, and W. A. Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In C. Sinz and U. Egly, editors, *Theory and Applications of Satisfiability Testing - SAT 2014*, LNCS 8561, 422–429. Springer, 2014.