A Distributed and Trusted Web of Formal Proofs

Dale Miller

A distributed and trusted web of formal proofs

Dale Miller

Inria & LIX, École Polytechnique
Palaiseau, France

Abstract. Most computer checked proofs are tied to the particular technology of a prover's software. While sharing results between proof assistants is a recognized and desirable goal, the current organization of theorem proving tools makes such sharing an exception instead of the rule. In this talk, I argue that we need to turn the current architecture of proof assistants and formal proofs inside-out. That is, instead of having a few mature theorem provers include within them their formally checked theorems and proofs, I propose that proof assistants should sit on the edge of a web of formal proofs and that proof assistant should be exporting their proofs so that they can exist independently of any theorem prover. While it is necessary to maintain the dependencies between definitions, theories, and theorems, no explicit library structure should be imposed on this web of formal proofs. Thus a theorem and its proofs should not necessarily be located at a particular URL or within a particular prover's library. While the world of symbolic logic and proof theory certainly allows for proofs to be seen as global and permanent objects, there is a lot of research and engineering work that is needed to make this possible. I describe some of the required research and development that must be done to achieve this goal.

1 Introduction

A great triumph of the World Wide Web is the ease at which anyone can access a great deal of diverse information. Such information spans a significant portion of human activity, ranging from information that is continuously updating (e.g., traffic information, flight schedules), to personal data (e.g., social networks, home security), to academics (e.g., research papers, libraries). A glaring flaw of the world wide web is that it provides few tools to help consumers of information actually trust the assertions that may be made in documents that are retrieved from the web. While techniques such as digital signatures can be used to determine the author of signed information and blockchains can be used to manage the provenance and timing of sources of information, few techniques are available to provide trust in what is actually claimed by information sources.

In many cases, we trust certain information sources since we trust the reputations of the corporations or individuals that generate and distribute that information. However, trust based on reputation has serious problems since those agents may have their own reasons to make their assertions and this may have little to do with truth: e.g., there is no guarantee that past behavior of an entity forces continued trustworthy behaviors; entities may be wrong about the assertions they make; and many bad actors (e.g., criminals or nation-states) are ubiquitous in our modern networks and attempt to undermine reputable sources. Of course, trusting various agents is critical since trust leads to beliefs, and these can lead to a willingness to take action. For example, by trusting a particular engineering company, you might be willing to fly on the planes that they make. Similarly, trusting in the secure implementation of encryption software can lead one to use such software to communicate financial transactions.

Since so many of our actions today are informed by or rely on electronic and web-based services, trust in those services is a fundamental, societal need. Unfortunately, the news is full of reasons why we should, in fact, not trust our currently designed web-based systems. In particular, there are routine attacks on our information systems using computer viruses, malware, fake news, monitoring back doors, and phishing attacks. Such attacks cause a profound erosion of trust in our communication infrastructure and in the proper functioning of the many institutions (e.g., banks, governments, hospitals, etc.) that operate via that electronic infrastructure.

There are two significant and well-understood sources of permanent trust that have not yet been applied in any significant way to modern electronic infrastructure. One involves the notion of formal (mathematical) proof, and the other is the notion of reproducibility (replication) that is a critical component of the scientific method.

The first anchor for trust: formal proofs Formal proofs have helped establish trust in two different epochs. In the late 1800s and early 1900s, there were various crises in the foundations of mathematics: the increasing use of both abstractions and infinitary methods were leading to inconsistencies and to a significant erosion of trust in some areas of mathematics [4]. The logicism project—dating back to Leibniz and continuing with Dedekind, Frege, Russell, and Whitehead—provided an approach to reducing mathematics to logic by insisting on using axioms, inference rules, and formal proofs. One may choose different sets of axioms (e.g., with or without the axiom of excluded middle or of choice) but the existence of a formal proof based on those axioms is a source of trust in mathematics. In the late 1900s and early 2000s, there have been numerous crises in the digital infrastructure used by society: the increasing use of computer systems to operate and control much of society's infrastructure has lead to serious system failures (e.g., Ariane 5, the 2015 Ukrainian power grid attack), data breaches, etc. Such incidents seriously erode the trust in our digital infrastructure and in the institutions built on it. The general topic of *formal methods* has been introduced and studied in order to provide some guarantees about some aspects of computer systems: most aspects of formal methods can be related to logic and formal proofs.[1] Formal proofs about the correctness (or partial correctness) of computer hardware and software can greatly help us trust

[1] Even the highly successful formal method of *model checking* can be given a foundation using formal proofs [27].

such hardware and software. While it is difficult to formally prove the full, functional correctness of complex hardware and software systems, it is possible to provide formal proofs of some aspects of such systems: for example, while it might be nearly impossible to prove that a video codex plugin correctly displays video, security concerns might be addressed by proving that that plugin touches only narrowly constrained regions of memory (interpreters of binary data streams are typical places where software errors—particularly buffer overflows—can lead to exploits).

In this paper, we shall focus on formal proof and on how they might be shared and used. However, as we now observe, trust in formal proof is intimately tied to trust in proof checkers, which forces us to consider a second source of trust.

The second anchor for trust: reproducibility The use of formal proofs as a font of trust faces, however, at least one serious challenge: how do you trust a formal proof? Except for toy examples, formal proofs are generated by computers and can only be checked by computers. Thus, it seems that we must necessarily trust in the correctness of proof checkers, which rely on the processors, operating systems, libraries, compilers, and linkers used to implement them [38]. Since our universal experience with computer programs is that they can be riddled with errors, we can easily doubt proof checkers and, consequently, doubt whether or not a given 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 need to explicitly invoke the second, foundational approach to trust, which we take directly from the *scientific method*: namely, proof checking needs to be *reproducible* in the sense that it is an activity that any skeptic should be able to undertake.

The seal of the Royal Society enshrines the creed "Nullius in verba" (take no one's word for it): that is, before trusting something, check it for yourself. The goal of this paper is to design an overlay for the World Wide Web that will enable the emergence of open, transparent, permanent, growing, and flexible collections of assertions and theorems. Both of these anchors for trust will be exploited in our design.

After surveying the state-of-the-art in Section 2, we briefly discuss the interplay between formal proof and trust in Section 3. In Section 4, we break down the problem of building a distributed and trusted web of formal proofs into five challenges, each of which is discussed in the next five sections. Finally, in Section 10, we describe some of the possible consequences of deploying the infrastructure described in those challenges.

2 State-of-the-art

The proposal in this paper involves building an overlay on the internet that can be used to distribute formal proofs. At one level, of course, a formal proof is a document and such documents can be hosted and moved around the World

Wide Web as if they were any other document, with extensions such as `html` or `pdf`. However, formal proofs do have additional properties that suggest that such an overlay can and should have additional structure. Before we describe this new structure for formal proofs, we first overview several aspects of the current state-of-the-art in the following topics: the World Wide Web, formal proof, and the modeling of trust.

2.1 The World Wide Web

In the matter of a couple of decades, the World Wide Web changed everything about how electronic documents were produced and shared. By designing some protocols and standards (e.g., `HTTP` and `HTML`) as well as some tools (e.g., web browsers), the sharing of documents created a revolution in the way people access information. We point out two lessons learned from the development of the World Wide Web that are particularly relevant to us here.

Standardization versus emergence One of the strengths of the early web was that it developed some standards, but it did not standardize too much. Instead, many features that we now see as integral to our use of the web—ranging from curated sites like Wikipedia to programmable web content based on JavaScript—were left to evolve along their own trajectory. Thus a goal of early standards should be to allow for a diversity of new structures to *emerge* later.

Moving from a cooperative to an adversarial environment In the beginning, most users of the web were academics who were mainly using this distributed information system in a *cooperative* fashion. If someone found a bug in a protocol or implementation, that bug was reported so it could be fixed; back-doors existed to allow for testing; and information sources such as academic papers, user manuals, etc., communicated true and fact-based information. A decade after the invention of the world wide web, however, we have seen that the web must now defend against adversarial behaviors. Now, if someone finds a bug in a protocol or implementation, that bug can be sold in a market to people interested in exploiting it; back-doors are placed in systems to allow bad actors to infiltrate or remotely monitor those systems; information sources can be lies meant to manipulate voters. Many new techniques—many of them cryptographic in nature—have been invented and deployed in order to provide for authentication, privacy, and transparency.

2.2 Formal proof

Completely formal proofs of significant theorems are produced by machines and are checked by machines: they are not meant for direct human consumption. Occasionally, a human can create and/or read a formal proof but that is the exception: formal proofs are meant for machines. Similarly, calculations can be done without machines, but far fewer would be done, and it would be hard to trust a human to do the correct sequence of steps every time. In the adversarial world of the World Wide Web, formal proofs can also be used to guarantee that certain classes of errors do not occur in software and hardware systems: sometimes, it is possible to declare a system to be *hacker-proof* [42].

Discovering and building formal proofs has often been considered a source of trust in both mathematics—as argued by V. Voevodsky [45]—and software systems—which is a theme of D. MacKenzie's book *Mechanizing Proof: Computing, Risk, and Trust* [31]. We describe here a few examples of how formal proofs and proof checking have been or are currently being used, supported, and standardized.

Proof-carrying code The proof-carrying code (PCC) project [35] is an important precursor for our goals in this paper. The PCC project stressed that trust in programs distributed as byte code could be based on actual formal properties of the distributed file and not on the authority which may have signed it. While the PCC project targeted a narrow application (mobile code for insertion into operating system kernels) and had a narrow perspective on proofs (dependently typed λ-terms and oracle strings [36]), its use of formal proofs was, in large part, independent of the technology that created them: as such, their proof certificates could be checked by third-party checkers, which results in greater trustworthiness. Various follow-on efforts were also pursued where formal proofs and proof checking was attached to authentication [8], access control [12], and a file system [24].

Frameworks for logics and proofs The exact nature of what is accepted as formal proofs has been a long-standing debate, and it has spawned lots of work on *logical frameworks* [37] as well as *mathematical knowledge management* systems and frameworks [20]. Clearly, certain well understood and flexible logics should be accepted by any framework: these include, for example, first-order classical and intuitionistic logics. Various additional axioms might well be added, accounting for set theory, arithmetic (Peano or Heyting), category theory, and even various type theories. Various people have built frameworks for expressing many logics in an ecumenical fashion [18, 25, 30, 40]. The *QED manifesto* [6, 26] described the value of formalizing a great deal of mathematics and proposed a vision in which the world's many theorem provers would work together to help construct that formalization.

Specialized and general-purpose proof certificates There have been a few attempts at defining computer-based representations of proofs that are output from theorem provers (automatic or interactive) for the expressed purpose of being checked by an independent proof checker. Such representations of proofs are often referred to as *proof certificates*: that is, documents that contain sufficient details for the complete construction of a formal proof. The SAT solver and term rewriting communities have developed special-purpose certificates for particular classes of theorems: DRUP/DRAT for SAT solvers [46] and CPF for term rewriting [23]. The Dedukti proof checker [10, 11] targets provers working in intuitionistic arithmetic while the *Foundational Proof Certificate* (FPC) framework [21] targets more generally classical and intuitionistic logic and arithmetic.

2.3 Modeling trust

As we shall discuss more in Section 3, a basic notion of *trust* is intertwined with formal proof. We mention here some topics related to trust below.

Chain-of-trust An important mechanism for ensuring trust in today's digital infrastructure uses the *chain-of-trust* ideas: here, a trusted principal (named by a *root certificate*) is used to anchor and to sign other principals that it deems as trustworthy. Two such chains-of-trust in use in systems today are the *public key infrastructure* (PKI) (used to secure web browser sessions) and the UEFI Secure Boot (used to secure the loading of trusted firmware and operating system components on computers). Although such an approach to trust is fundamental in today's world wide web architecture, this approach is based not on the content of signed documents but on reputation: e.g., in the UEFI secure boot framework, one trusts binaries not because we know them to be "safe" but because Microsoft has signed them.

Modal logics for modeling trust relations It is possible to model more sophisticated organizations of trust relationships than those simply given by chain-like constructions. In particular, one might be interested in hypothetically trusting a principal and then see what flows from it. Such reasoning is required for the modular construction of software that could be part of a chain-of-trust. In fact, the notion that proof checking should be reproducible means that one presumably only trusts a theorem if *several* independent proof checkers are able to certify it: no one "root" proof checker is meant to be universally trusted. Modal logics, such as access control logic [2] and authorization logic [41], have been used to model such trusting relationships in a number of computer science settings.

Trust management systems While the most basic notion of trust is central here, more general questions about how to manage trust will be left for elaboration in later work. We shall use formal method techniques as a way to *initiate* trust: this is in contrast to the work by, say Carbone, Nielsen, and Sassone [19], where formal methods are used to *manage* trust. We shall also not consider questions such as the *degree* of trust or whether or not trust is transitivity [39].

3 Formal proof and trust

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 proofs, particularly, mathematical proofs. Humans (e.g., reviewers) are often used as proof checkers. We have seen in recent years, important limits to what humans are capable of doing with proofs. Voevodsky [45] documents errors that were found in a series of mathematical papers. Also, many proofs are massive in size and complexity, and having humans check such proofs for correctness seems both impossible and counterproductive (would you really trust a group of humans to get every detail checked carefully in such large, technical proofs?).

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 achieved by humans. Formal proofs have also been built for mathematical theorems that humans would find nearly impossible to check carefully (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 our 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-style 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. As we argued in Section 1, the reproducibility of proof checking is required. While proof checking depends on technology, it should not be dependent on any specific technology. Fifty 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 will require 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 described in multiple ways, including using model theory and using a variety of proof structures. Furthermore, there are numerous papers about and implementations of the basic algorithms that underlie proof search and proof checking. It is easy to find notions of formalized reasoning that are not 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 structure 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 standardized proof certificate formats than their search programs can output. These formats have names such as DRUP [28] and DRAT [46], 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 [43] that can be used to check whether or not a given rewrite system is terminating or confluent. The logical framework LF [25, 37] has been used by a number of projects as a general setting for storing and certifying proof structures [7, 8, 12, 24, 35, 48].

In a distributed world of formal proofs, one must expect there to be a great diversity of proof systems that would be tied to different theorem provers, and those proof would like all have different structures. If we need to trust a large number of different proof checkers in order to treat a large number of proof formats, it is hard to see how we have improved trust in proof checking. Presently, there are at least two broad-spectrum proof certificates formats that are being developed. The Dedukti system [10, 11] mixes functional programming style rewriting with dependently typed λ-calculus to provide a proof certificate for an increasing number of theorem provers working in higher-order intuitionistic logic. The *foundational proof certificates* framework [21, 33] uses 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: skeptics 50 years from now will be able to read the associated literature, build their own proof checker, and recheck any proof certificates written in these technology-independent formats.

4 A web of proofs and its challenges

A perspective on how to organize a global and distributed collection of formal proofs is given in Figure 1. Several attributes of web-style distribution are listed on the left of that figure: these attributes are divided into two sets of rows. The first set of rows deals with infrastructure, and the second set of rows deals with emergent structures. The middle column describes the values given to those attributes when one considers the standard world wide web of documents. Hopefully, all the items listed in that column are familiar to the reader. The third column imagines how those attributes are changed in order to provide a possible treatment of a global and distributed collection of formal proofs. Our main goal here is to focus on describing an infrastructure that could support a global, distributed, and trustworthy treatment of formal proofs.

We now list five challenges that seem appropriate for attacking this problem of a global and distributed web of formal proofs.

Challenge 1: Permanent electronic documents. This challenge deals with supporting the signing of almost any kind of assertion and with making such a signed file globally available. A wide range of assertions should be treatable

Web	**of documents**	**of formal proofs**
Standards and infrastructure		
Documents	Files in various formats	Proofs in various formats
Standards	SGML, HTML, XML, etc	FPC, Dedukti, CPF, DRUP, etc
Naming	URI, DOI	Content addressable storage (CAS)
Transport	HTTP, FTP	In addition: IPFS
Trust	certificate authorities, open source code, encryption, public logs, etc	In addition: *reputation* (e.g., signed by Coq 8.1) and *reproducibility* (rechecking proof evidence)
Emergent structures		
Access	browsers, JavaScript	proof browsers, interaction with proofs
Curation	Wikipedia, etc	proof libraries, textbooks

Fig. 1. Comparing the World Wide Web for documents with what is proposed for proofs

by any implementation of this challenge: for example, a news article (written in a natural language) could be signed by its author, and that article and its signature should be available universally. In that general setting, assertions are not assumed to be logical expressions, and formal proofs may not exist. None-the-less, it might be possible to have trust in such signed texts.

Challenge 2: Structuring libraries of theorems and their proofs This challenge commits to representing assertions as logical expressions and addresses the issue of assembling and structuring collections of proved formulas into theories and libraries. The infrastructure underlying this challenge can be deployed in a setting where formal proofs are absent or where their structure is not used.

Challenge 3: Operating on proofs This challenge proposes to develop tools and services—beyond proof checking—that can be applied to a range of proofs. This challenge requires working with details about the structure of formal proofs.

Challenge 4: Replication in experimental science This challenge takes a particular aspect of formal proof, namely the direct specification of computation, and applies it to the problem of describing clearly the data and computations that are used within research in experimental mathematics and science. Formal proof checking could then be used to directly check and certify at least some of the computational and argumentative judgments that go into forming scientific conclusions from scientific observations.

Challenge 5: Deployment This challenge considers the initial steps in attempting to deploy the framework we have proposed.

Each of these challenges will be addressed in the following five sections. These challenges have been sorted to reflect their dependency on the specifics of formal proofs. In particular, Challenge 1 needs little to no information about the structure of formal proofs, while Challenge 5 is specifically about implementing proof structure.

5 Challenge 1: Permanent electronic documents

Assume for the moment that we have suitable standards for capturing a formal proof in a computer file. Such a file will likely contain declarations of the constants and definitions and previously proved results that are needed in that proof. Since a foundational motivation for logic and proof is to be universal and technology-independent, such a file could exist outside of any specific library for a theorem prover or outside some specific tree structure based at some root URL. That is, a proof document could simply exist anywhere and everywhere. This leads us to the following task.

Task 1: Design a global and permanent collection of signed documents The usual approach to ensuring wide-spread access to a document is to locate them in a specific directory tree on a specific machine or web service using, say, a URL. Such a storage scheme has several problems: such machines or web services need to be trusted (so that they do not alter stored documents), and directory trees and servers often change over time. A different kind of storage mechanism which addresses those two issues is the so-called *content-addressable storage (CAS)* in which the hash of a file is its name. A start on an appropriate CAS facility can be found in the Interplanetary File System (IPFS) [13]. In such a system, one requests a file by using its hash. As a result, it is easy to confirm that the file that is retrieved from actually has the required hash: as a result, we can have a high assurance that the file was not modified during storage and transfer. Also, if that file is cached locally using its hash as its name, one never needs to return to the distribution system to check for a new version of the file: another version would, of course, have a different name (i.e., hash).[2]

Another key component of representing proof objects is the *assertion* that, in fact, a particular document contains a valid proof. It is the job of *proof checkers* to check a document and to assert that that document contains a valid proof of the proposed theorem (also contained in the proof document). The act of an agent claiming to have proved a theorem or checked a proof can be modeled using cryptographic signing techniques (e.g., using public keys). Signing is usually done by a human or organization (collectively called *principals*). In particular, a given proof checker does not sign a document with its private key: instead, it would need to sign it with the private key of the principal that executes that checker. Since there is no way to be certain that a proof checker is not manipulated, we really need to trust the operator of the proof checker to be doing the appropriate and trustworthy validation steps with a proof checker. A good starting point for such signing is employing the public key infrastructure of private/public asymmetric encryption: principals are identified with their public key, and any

[2] We can assume that hashes are computed on the text contained in files: we do not rely on the more sophisticated notions of homomorphic hashes [9].

assertions that principals make are contained within cryptographically signed documents.

Assume that a principal has encryption key K. When K signs an assertion, say, A, we write $[P\ \mathit{says}\ A]$. Such an assertion is another (small) document that should be added to the CAS facility so that it is, in principle, available to anyone. The fact that such signed assertions are globally available plays an important role in the transparency of this system: if one eventually finds out that a principal asserted that a non-theorem is provable, the willingness of people to trust that principal would be questioned.

For our purposes here, we consider the following three kinds of assertions. The assertion $\vdash B$ is simply the statement that the formula B is provable. The assertion $\Xi \vdash B$ is the statement that the proof certificate Ξ was checked and lead to a formal proof of B. Finally, we should allow various kinds of meta-data M to be included in signed assertions, for example, $M, \Xi \vdash B$: such meta-data could include the time of the checking, the version number of the software used to do the checking, etc. If I have a file that encodes the signed file $[P_1\ \mathit{says}\ \vdash B]$, do I trust that B is, in fact, a proper theorem? If I am skeptical, I could try to use a different theorem prover, say, P_2, and get it to prove B: in that case, the additional signed file $[P_2\ \mathit{says}\ A]$ would appear. Having two theorem provers declare a formula to be a theorem might, in many cases, provide enough evidence to trust their assertion. Since theorem proving is a difficult task in most domains, it would be much more useful to attempt to reproduce signed documents of the form $[P_1\ \mathit{says}\ \Xi \vdash B]$ since I could attempt to find another proof checker P_2 and use it to assert $[P_2\ \mathit{says}\ \Xi \vdash B]$. On the other hand, meta-data is likely to be information that cannot be rechecked, and one either chooses to trust it or not.

Task 2: Select a modal logic for tracking trust Critical to being able to use the global library of signed documents is the need to track principals, particularly, those which a given principal chooses to trust or not. I may choose to trust a particular authority: e.g., I might trust any theorem that I was able to prove using a particular version of the Coq theorem prover [15]. Clearly, one's rules for trusting some principals must be something that is made explicit and trackable. Within the general setting of epistemic logics for knowledge and belief [29], there are various modal logics that can be used for such tracking. The logics DCC (by Abadi [1]) and NAL (by Schneider et al. [41]) are candidates for initial development. The fundamental modal operator here will likely be $[P\ \mathit{says}\ A]$, meaning that principal P (identified by its public key) asserts the statement A. The global collection of signed documents provide precisely the semantics for that modal statement.

A consequence of Tasks 1 and 2 should be a permanent, transparent, and trackable infrastructure for publishing signed assertions. This framework will provide rudimentary support for trusting agents and the documents that they sign. Note that this infrastructure would work independently from the exact specification of formal proof.

6 Challenge 2: Structuring libraries of theorems and their proofs

Unlike most approaches to building proof assistants, we will view libraries of theorems and proofs as an *emergent* aspect of collecting theorems and proofs. In a similar fashion, Wikipedia is an emergent feature of the underlying structure of the basic protocols (URI, HTTP, HTML, etc.) of the web. Any number of people should be able to curate their own specific libraries or textbooks from the growing, dynamic collection of theorems and proofs. Andrews's commentary [5] on the QED manifesto [6] is worth repeating: many logics and proof systems should be allowed to exist, and future users of formalized proofs should be the ones to determine which logics and proof systems dominate (if any).

The dependency of one set of theorems to help establish another set of theorems can be captured (as a consequence of using CAS) as a Merkle tree, a structure that guarantees non-cyclic dependency [32]. A more crucial aspect of library building is the problem of *sharing*. When assembling a library (or a textbook), one wishes never to repeat the same theorem within a collection of theorems and one is willing to do significant work to ensure that as much proof structure and as many lemmas as possible are reused and not repeated. But libraries of theorems and proofs are not just *any* collection of theorems and proofs, and their large-scale structure needs to be understood.

Task 3: Understand the large scale structure of theories and proofs One of the first emergent structures that will appear is that of *theory*. Several theories—set theory, type theory, higher-order logic—have already emerged within the study of the logical foundations of mathematics. Most existing theorem provers have one of these theories built into their foundation. To be liberated from their technologies, such theories must be clearly identified, categorized, and referenced consistently.

An important test of a good structuring of signed assertions is the possibility of rich notions of *sharing*. By sharing, we mean more than simply making theorems and proofs available to others: rather, we also mean that we will need to provide means for proofs to be *reused*. Since the late 1960's and the start of de Bruijn's Automath system [17], it has been recognized that libraries in mathematics can be seen as large-scale λ-terms. When λ-terms are used in writing computer programs (as in Lisp or ML), identifying sharing is generally seen as not worth the effort. As we mentioned above, when assembling a library or textbook, one often commits a great deal of energy into making certain that the same theorem or proof is not repeated. The use of Merkle trees within the CAS setting makes reuse of textually equal documents a triviality: we are, of course, interested in making certain that semantically equal assertions are reused.

Task 4: Determine how to do binding in highly distributed proof libraries A key component of most sophisticated reasoning is the use of bound variables at the level of proof structures: these are called *eigenvariables* by Gentzen and λ-bound variables by most logical frameworks). While this concept of proof-level-binding is treated well in modern proof systems, when we need bindings to work in a highly distributed setting, an entirely new approach to such bindings must be taken. A good starting point for this kind of binding construction is the cryptographic constructions describe by Abadi, Fournet, and Gonthier in [3] and by Comon and Koutsos in [22].

These tasks will uncover the structure needed to organize large-scale permanent and trusted libraries of named documents. Some aspects of logic and proof—namely the need for uniquely naming assumptions and parameters—will be needed for here.

7 Challenge 3: Operating on proofs

By viewing proofs as structures with well-defined properties, a rich collection of services on formal proofs can be provided. Computing directly with formal proofs can greatly increase their value beyond the initial intent of certifying their correctness.

Task 5: Develop support for foundation-independent proofs During the past several decades, a number of different foundations for mathematical reasoning have been proposed and implemented. Foundations such as set theory, type theory, category theory, and higher-order logic have all played a role in anchoring mathematical reasoning to axioms and inference rules. In the choice of logic, there is also the familiar choice that needs to be made between classical and intuitionistic logics, particularly when dealing with infinity and undecidable propositions. In mathematical and computer science practice, however, there is a great deal of formal argumentation that can be done independent of exactly what foundations one picks: that is, much of mathematics can be established via "foundation-less proofs": for example, a proof in number theory of, say, Fermat's Little Theorem can be done without much regard to exactly which foundations one is using. Since such proofs are critical to preserve, we need to find ways to structure different proof environments to provide different ways to anchor such foundation-less proofs into different theories of foundations. It is also the case that the same theorem may have many different proofs and that these different proofs may have a range of different uses. For example, some proofs might be structured with interesting abstractions that make them easier for humans to read; other proofs might have a great deal of detail added to make them easier to check; still other proofs might be structured so that their constructive content is more apparent; and still other proofs might discard many details so that they become more like proof outlines that could also serve as proofs of related theorems.

This issue of "foundation-less proofs" is rather similar to the way software is often organized: computer programs are often organized around an application program interface (API). That is, an application program expects to use a particular interface with lower-level operating system features and is not expected to break the interface abstraction to code directly using low-level features of an operating system or run-time system. As a result of obeying that interface, the code could execute properly on many different hardware and operating systems.

Task 6: Demonstrate proof-checking-as-a-service The proof certificates produced by theorem proving systems generally pose trade-offs for proof checkers. Certificates can provide a great deal of detail, which makes them simple to check but makes them large and hard to reuse. On the other hand, they can be compact and contain just essential details: in that case, proof checking requires doing some proof reconstruction [34], which can be computationally expensive. In either case, proof checking can be a challenge for many computer systems to perform: this could be particularly true of mobile phones. Of course, some principal, say `Lib`, could play the role of curating only assertions that its own proof checker can check. When a third-party vendor, say `Vendor`, wants to sell an application to a mobile phone customer who has certain (published) security requirements, the vendor may contract and pay `Lib` to certify its formal proofs that its applications meet those security requirements. The mobile phone would only then need to check the existence of a signed assertion from `Lib` that it has checked that the application satisfies the required security requirements. In such a setup, if, for some reason, `Lib` is not considered trustworthy (such as the discovery that `Lib` had once signed a non-theorem), another proof checking principal could be substituted.

Task 7: Develop a range of operations on proofs Many valuable operations on proofs can be performed by proof checkers. For example, many proofs encode algorithms, and a proof checker can be used to execute those algorithms. For another example specific to the FPC style of proof certificate: during the process of checking an implicit proof certificate (one with few details), a proof checker must fill in all missing details, a step that usually requires sophisticated mechanisms such as unification and backtracking search [16]. The resulting detailed proof can then be check by a simple-to-write-and-trust proof checker, thereby alleviating the need to trust the more sophisticated proof checker. Distilling information from a proof (the converse of elaboration) is also an important service that proof checkers can provide: for example, distilling can extract the set of assumptions used in a proof, the number of times such assumptions have been used, the instances at which certain assumptions are used, etc.

Task 8: Develop tools for interacting with proofs It is the rare formal proof that can be printed in such a way that beginners and experts can understand it. It seems much more likely, however, that people could learn from a formal proof by *interacting* with it via a *proof browser*: such a tool would allow a proof to be explored incrementally and partially. For example, someone might learn a lot from a proof if they could analyze only one of many cases, or use the proof to compute a specific witness given a user-provided input. Since proof checkers written using the FPC framework [21] are capable of elaborating proof certificates into fully detailed sequent calculus proofs, the rich meta-theory surrounding sequent calculus (in particular, the cut-elimination theorem) is available to provide for such dynamic and rich interactions.

8 Challenge 4: Replication in experimental science

In their most general setting, theorems and proofs can include both data and programs. Capturing reasoning steps as well as data and programs, is a major challenge today in what is often called the *replication crisis.* A treatment of formal proofs must necessarily provide a setting where this problem is addressed. Treating formal proofs is rather general and possibly naive, but it will provide a framework that can be elaborated and optimized. Central to the application of this framework to the experimental sciences is the ability to describe computations and data as parts of proofs (a concept familiar to such theorem provers as Coq, Agda, and Isabelle) as well as support novel kinds of inference such as those used in statistical reasoning. The modern view of *formal proof* makes it possible to incorporate all these aspects of reasoning in such scientific domains.

The *replication crisis*, a term coined in the early 2010s, arose when many researchers found it difficult or impossible to replicate the published results in areas of psychology, medicine, and computational and experimental mathematics [44]. Since replication of results is a cornerstone of our trust in science, the trustworthiness of these scientific areas has been called into question. Of the various methods for addressing this crisis, the one that is most directly related involves encouraging researchers to publish their experimental data and their software for analysis along with their papers announcing their results. For example, the Open Science Framework (osf.io) and the Life Sciences Protocol Repository (www.protocols.io) provide a platform for scientists to collaborate in an open setting and where various (proprietary) technologies (Dropbox, Google Drive, etc.) are linkable. While such a framework provides much-needed transparency, no technology-independent definition of computation and reasoning steps are provided.

Task 9: Address the replication problem in experimental science Both the terms "reproducibility" and "replication" are used in the literature: we use the latter term here to mean redoing an experiment and gather new data; the former term will be targeted at rechecking the computations and arguments made on the results of an experiment. The modern view of formal proof is so general that it encompasses the specification of data and programs. The inference rules commonly used in scientific reasoning involving experimentation—for example, probabilistic and statistical reasoning—would need to be captured as inference rules in (mathematical) logic. In the end, the formal documents that are proofs can serve a dual purpose: they can be executed and validated by the many proof checkers we will have available, and they can serve as a simple, clear, and precise descriptions of data, computations, and reasoning steps. Given their formal encoding, such calculations and reasoning steps can be reproduced by others, even without reference to the large machinery of proof into which they sit.

Our formal proof setting will necessarily provide a transparent and permanent means of communicating the results, computations, and reasoning steps that represent the analysis of scientific experiments. As a result, this framework should, in principle, provide a formal approach to replication in the scientific setting.

9 Challenge 5: Deployment

A major challenge to this project is to make it possible for implementers and developers of theorem provers to export their proofs and libraries so independent systems can check them and import proofs from the global and permanent collection of checked assertions. We consider just one task associated with deployment.

Task 10: Build a prover that simply links other proofs Design and build a theorem prover that does not have significant proof search methods of its own: its main job is to link proofs (via an established white-list of trusted agents) and make conclusions. Such a prover could be used to build large outlines in a distributed fashion. We might allow an agent, called "wild-guess" to sign a formula and, if we allow that agent to be trusted (at least temporarily), then we might be able to complete a large scale proof. Of course, one eventually wants to have additional proofs added so that anything signed by wild-guess is eliminated. Tracking provenance is central here.

10 Planned versus emergent structures

The world wide web started with the deployment of a few extensions to internet protocols (e.g., the transport protocol `HTTP` and the document format `HTML`) and a few tools (e.g., browsers such as NCSA Mosaic and Netscape Navigator). It is a remarkable feature of the web that so many new structures have emerged that exploit the original standards (and their descendants). If Tim Berners-Lee had over-designed the early web systems so that it was just intended to be a large distributed hypertext library of research-oriented results and papers, then there would probably have been a significant delay in building what we now know as the web, especially since only a small community would have initially been interested in that particular application. By leaving much of the framework open, a great diversity of people and projects—far from providing a library of academic results—was able to flourish.

Linking formal proof and trust is not a new theme, of course. For example, it is part of the vision of the *semantic web* [14]. Two prior efforts at doing this over-specified their target application areas (at least in hindsight) and, as a result, their potential as a framework was limited. For example, the work on Proof-Carrying Code (PCC) [35], mentioned in Section 2.2, used formal proof to deal with trust but limited the application areas to the execution of untrusted code on mobile devices. Similarly, the 1984 QED manifesto [6, 26] proposed the construction of a computer-based database of all mathematical knowledge, along with checked, formalized proofs. The intended audience for this manifesto was "working mathematicians". One of the reasons offered by F. Wiedijk [47] for why this project failed in this particular goal was that there is currently no compelling application for fully mechanized mathematics among working mathematicians. (As mentioned in Section 3, V. Voevodsky [45] is a recent counterexample, since he was a working mathematician with a compelling reason to mechanize parts of mathematics.)

Instead of over-designing, we should let things emerge (given that the right infrastructure is in place). Calling out the QED manifesto again, we can illustrate two examples of over-designing.

1. QED blended the need for formal proofs with concerns about the human readability of formal proofs. In contrast, this proposal makes the simplifying assumption that formal proofs are meant for machines. Human reading and understanding of formal proofs are left as an activity that should emerge (see Section 7). Also, having all proofs understood by humans in a "working mathematics" setting might be appropriate, it is certainly not desirable in a more general setting. For example, the proof that a specific device driver never touches memory outside certain boundaries might be extremely important to have in some settings, but one would not necessarily expect a human to read, check, or understand that proof.
2. The topics that get formalized within this framework should be expected to arise organically, from compelling community needs. At the end of the QED manifesto, however, the suggestion was that it might be good to work on ring theory. But it is hard to know what topics are compelling. For example, one might make the argument that formal proofs about cryptology might be an important starting point for building trust in our digital world. Such a focus on that might then lead organically to a need to formalize parts of number theory. The importance of ring theory might eventually arise, for example, as a useful way to organize some of the theorems and proofs surrounding number theory.

11 Concluding remarks

In the areas of mathematics and software correctness, we might be tempted to say that trust begins with a formal proof. But the reality is that formal proofs are not known to be correct without the execution of a proof checker. Thus, trust must also be based on the correct execution of a possibly complex software and hardware system: that is, trust must also be based on *reputation* (in this case, on the computer system checking a proof). Thus, our starting point is that assertions are signed by a principal (an individual or an institution). Rudimentary forms of trust are then possible by simply providing a white-list of principals whom we may trust. While an approach to trust-based only on the reputation of principals is a requirement in the grand scheme, it also has serious weaknesses. A stronger form of trust arises by examining any *evidence of proof* (e.g., a formal proof or a proof certificate which can be elaborated into a formal proof), since it is possible for skeptics to write their own proof checker and check that the assertion is, in fact, a valid proof. Trust can thus be based on the *reproducibility* of checking.

Given this interplay between formal proof and trust, we have described a setting in which formal proofs can be distributed and trusted in a web-like network. This proposal attempts to turn the world of theorem proving inside-out: the provers that now form the center of the world for their users will be placed at the periphery of a web that will be focused on accumulating and reusing the results of those many provers. As a result, this framework should support any number of theorem provers since they only need to be able to export their proofs in a way that some trusted proof checkers can certify. In the process of reshaping that world, we needed to address also a number of issues—principals signing assertions, transparency, and global access—that provides an integrated infrastructure that might also be used in journalism as well as experimental sciences.

References

1. Abadi, M.: Access control in a core calculus of dependency. Electr. Notes Theor. Comput. Sci **172**, 5–31 (2007)
2. Abadi, M.: Variations in access control logic. In: van der Meyden, R., van der Torre, L.W.N. (eds.) Deontic Logic in Computer Science, 9th International Conference, DEON 2008, Luxembourg, Luxembourg, July 15-18, 2008. Proceedings. LNCS 5076, pp. 96–109. Springer (2008)
3. Abadi, M., Fournet, C., Gonthier, G.: Secure implementation of channel abstractions. Information and Computation **174**(1), 37–83 (2002)
4. Alexander, A.: Infinitesimal: How a dangerous mathematical theory shaped the modern world. Oneworld Publications (2014)
5. Andrews, P.B.: Accept diversity (Aug 1994), email message archived at `http://mizar.org/qed/mail-archive/volume-2/0199.html`
6. Anonymous: The QED manifesto. In: Bundy, A. (ed.) 12th International Conference on Automated Deduction. pp. 238–251. No. 814 in LNAI, Springer-Verlag, Nancy, France (June 1994)
7. Appel, A.W.: Foundational proof-carrying code. In: 16th Symp. on Logic in Computer Science. pp. 247–258 (2001)
8. Appel, A.W., Felten, E.W.: Proof-carrying authentication. In: Proceedings of the 6th ACM Conference on Computer and Communications Security. pp. 52–62. ACM (1999)
9. Armknecht, F., Boyd, C., Carr, C., Gjøsteen, K., Jäschke, A., Reuter, C.A., Strand, M.: A guide to fully homomorphic encryption. Cryptology ePrint Archive, Report 2015/1192 (2015), `https://eprint.iacr.org/2015/1192`
10. Assaf, A., Burel, G.: Translating HOL to Dedukti. In: Kaliszyk, C., Paskevich, A. (eds.) Proceedings of the Fourth Workshop on Proof eXchange for Theorem Proving, PxTP 2015, Berlin, Germany, August 2-3, 2015. EPTCS, vol. 186, pp. 74–88 (2015), `http://dx.doi.org/10.4204/EPTCS.186`
11. Assaf, A., Burel, G., Cauderlier, R., Delahaye, D., Dowek, G., Dubois, C., Gilbert, F., Halmagrand, P., Hermant, O., Saillard, R.: 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)

12. Bauer, L.: Access Control For The Web Via. Ph.D. thesis, Princeton University (Sep 30 2003), `http://www.ece.cmu.edu/ lbauer/papers/thesis.pdf`
13. Benet, J.: IPFS-content addressed, versioned, P2P file system (2014)
14. Berners-Lee, T.: Semantic Web road map. Tech. rep., W3C Design Issues (1998), `http://www.w3.org/DesignIssues/Semantic.html`
15. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science, Springer (2004), `http://www.labri.fr/publications/l3a/2004/BC04`
16. Blanco, R., Chihani, Z., Miller, D.: Translating between implicit and explicit versions of proof. In: de Moura, L. (ed.) CADE 26 – International Conference on Automated Deduction, Gothenburg, Sweden, 2017. LNCS 10395, pp. 255–273. Springer (2017). `https://doi.org/10.1007/978-3-319-63046-5_16`
17. de Bruijn, N.G.: A survey of the project AUTOMATH. In: Seldin, J.P., Hindley, R. (eds.) To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism, pp. 589–606. Academic Press, New York (1980)
18. de Bruijn, N.G.: A plea for weaker frameworks. In: Huet, G., Plotkin, G. (eds.) Logical Frameworks. pp. 40–67. Cambridge University Press (1991)
19. Carbone, M., Nielsen, M., Sassone, V.: A formal model for trust in dynamic networks. In: SEFM. p. 54. IEEE Computer Society (2003)
20. Carette, J., Farmer, W.M.: A review of mathematical knowledge management. In: Carette, J., Dixon, L., Coen, C.S., Watt, S.M. (eds.) Intelligent Computer Mathematics, 16th Symposium, Calculemus 2009, 8th International Conference, MKM 2009, Grand Bend, Canada, July 6-12, 2009. LNCS 5625, pp. 233–246. Springer.
21. Chihani, Z., Miller, D., Renaud, F.: A semantic framework for proof evidence. J. of Automated Reasoning **59**(3), 287–330 (2017). `https://doi.org/10.1007/s10817-016-9380-6`
22. Comon, H., Koutsos, A.: Formal computational unlinkability proofs of RFID protocols. In: Computer Security Foundations Symposium (CSF), 2017 IEEE 30th. pp. 100–114. IEEE (2017)
23. Certification problem format. `http://cl-informatik.uibk.ac.at/software/cpf/` (2015)
24. Garg, D., Pfenning, F.: A proof-carrying file system. In: 2010 IEEE Symposium on Security and Privacy. pp. 349–364. IEEE (2010)
25. Harper, R., Honsell, F., Plotkin, G.: A framework for defining logics. Journal of the ACM **40**(1), 143–184 (1993)
26. Harrison, J., Urban, J., Wiedijk, F.: Preface: Twenty years of the QED manifesto. J. Formalized Reasoning **9**(1), 1–2 (2016)
27. Heath, Q., Miller, D.: A proof theory for model checking. J. of Automated Reasoning **63**(4), 857–885 (2019). `https://doi.org/10.1007/s10817-018-9475-3`
28. Heule, M., Jr., W.A.H., Wetzler, N.: Expressing symmetry breaking in DRAT proofs. In: Felty, A.P., Middeldorp, A. (eds.) Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings. LNCS 9195, pp. 591–606. Springer (2015)
29. Hintikka, J.: Knowledge and Belief: An Introduction into the logic of the two notions. Cornell University Press, Ithaca (1962)
30. Kohlhase, M., Rabe, F.: QED reloaded: Towards a pluralistic formal library of mathematical knowledge. J. Formalized Reasoning **9**(1), 201–234 (2016)
31. MacKenzie, D.: Mechanizing Proof. MIT Press (2001)
32. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) Advances in Cryptology—CRYPTO '87. LNCS 293, pp. 369–378. Springer-Verlag, 1988 (16–20 Aug 1987)

33. Miller, D.: A proposal for broad spectrum proof certificates. In: Jouannaud, J.P., Shao, Z. (eds.) CPP: First International Conference on Certified Programs and Proofs. LNCS 7086, pp. 54–69 (2011). https://doi.org/10.1007/978-3-642-25379-9_6.
34. Miller, D.: Proof checking and logic programming. Formal Aspects of Computing **29**(3), 383–399 (2017). https://doi.org/10.1007/s00165-016-0393-z.
35. Necula, G.C.: Proof-carrying code. In: Conference Record of the 24th Symposium on Principles of Programming Languages 97. pp. 106–119. ACM Press, Paris, France (1997)
36. Necula, G.C., Rahul, S.P.: Oracle-based checking of untrusted software. In: Hankin, C., Schmidt, D. (eds.) 28th ACM Symp. on Principles of Programming Languages. pp. 142–154 (2001)
37. Pfenning, F.: Logical frameworks. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning (in 2 volumes), pp. 1063–1147. Elsevier and MIT Press (2001)
38. Pollack, R.: How to believe a machine-checked proof. In: Sambin, G., Smith, J. (eds.) Twenty Five Years of Constructive Type Theory. Oxford University Press (1998)
39. Primiero, G., Raimondi, F.: A typed natural deduction calculus to reason about secure trust. In: Miri, A., Hengartner, U., Huang, N.F., Jøsang, A., García-Alfaro, J. (eds.) Twelfth Annual International Conference on Privacy, Security and Trust, Toronto, ON, Canada, July 23-24, 2014. pp. 379–382.
40. Rabe, F.: How to identify, translate and combine logics? J. of Logic and Computation **27**(6), 1753–1798 (2017)
41. Schneider, F.B., Walsh, K., Sirer, E.G.: Nexus Authorization Logic (NAL): Design rationale and applications. ACM Transactions on Information and System Security **14**(1), 8:1–8:28 (May 2011). https://doi.org/https://doi.org/10.1145/1952982.1952990
42. Shein, E.: Hacker-proof coding. Commun. ACM **60**(8), 12–14 (Jul 2017). https://doi.org/10.1145/3105423.
43. Sternagel, C., Thiemann, R.: The certification problem format. In: Proceedings UITP 2014. pp. 61–72 (Oct 2014). https://doi.org/10.4204/EPTCS.167.8.
44. Stodden, V., Bailey, D.H., Borwein, J., LeVeque, R.J., Rider, W., Stein, W.: Setting the default to reproducible: Reproducibility in computational and experimental mathematics (Feb 2013), http://www.davidhbailey.com/dhbpapers/icerm-report.pdf.
45. Voevodsky, V.: Univalent foundations. Talk given at the Institute for Advanced Study (March 2014), http://www.math.ias.edu/vladimir/sites/math.ias.edu.vladimir/files/2014_IAS.pdf
46. Wetzler, N., Heule, M.J.H., Hunt, J.W.A.: DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In: Sinz, C., Egly, U. (eds.) Theory and Applications of Satisfiability Testing – SAT 2014, LNCS 8561, pp. 422–429. Springer (2014). https://doi.org/10.1007/978-3-319-09284-3_31
47. Wiedijk, F.: The QED manifesto revisited. Studies in Logic, Grammar and Rhetoric **10**(23), 121–133 (2007)
48. Wu, D., Appel, A.W., Stump, A.: Foundational proof checkers with small witnesses. In: Miller, D. (ed.) PPDP '03: Proceedings of the 5th ACM SIGPLAN international conference on Principles and practice of declaritive programming. pp. 264–274. ACM, New York, NY, USA (2003)