A distributed and trusted web of formal proofs

Dale Miller

Inria Saclay & LIX, École Polytechnique
Palaiseau, France

16th ICDCIT 2020:
International Conference on Distributed Computing and Internet Technology

9-12 January 2020, Bhubaneswar, India

A web of distrust

A great triumph of the World Wide Web is the ease at which anyone can access a great deal of diverse information.

A glaring flaw of the WWW is the lack of tools to help consumers of information actually trust the assertions made in documents.

Trusting is important since trust leads to actions.

- If I trust an particular engineering company, I fly their planes.
- If I trust Microsoft, I do my taxes on their computers.
- If I trust that ConjectureA is really a theorem, I will spend the next several months trying to prove ConjectureB.

Formal proof can lead to trust

Formal proofs have helped to establish trust during two different epochs.

In the late 1800s and early 1900s, there were various crises in mathematics.

- The uses of infinity and infinitesimals was questionable.
- Foundations were naive.

In the late 1900s and early 2000s, there have been numerous crises in our digital infrastructure.

- The use of error-prone computer systems to operate and control our infrastructure.
- Correctness, security, and privacy are often overlooked.

Formal proofs have improved the situation in both settings.

Advances in formal proof

In the area of mathematics,

- axiomatics of foundations (e.g., set theory) and
- new definitions of infinity (Cantor) and limit ($\epsilon - \delta$ arguments)

lead the way to greater confidence. There is still reason to worry: see, Voevodsky's 2010 and 2014 talks at the IAS.

In the area of computer science,

- proofs of functional correctness (c.f., Hoare, Floyd, etc),
- program testing,
- model checking, and
- better program language design

have lead to more confidence in the actual correctness of programs. There are still many reasons to worry.

Proof assistants help humans build formal proofs

Example theorems:

- $\sqrt{2}$ is irrational; the prime number theorem;
- the four color theorem;
- a C-compiler is correct (CompCert project, X. Leroy, Inria);
- a video codex does not reach outside its allotted memory.

Many existing systems: Coq, Agda, Isabelle, Lean, Abella, etc.

Proofs built in those assistants are specific to those systems: they are even sensitive to the assistant's version number.

Proof-checking needs technology but it should be independent of specific technology.

Formal proofs are for machines

Except for toy examples, formal proofs are produced by machines and consumed (checked) by machines.

Despite de Bruijn's pleas for weak frameworks [1991], *proof checkers* are complex, computational systems containing

- printers and parsers,
- interpreters and compilers,
- garbage collectors, and
- hardware processors.

All of these can have flaws. We have good reasons to be skeptical of proof checkers.

It is usually the *reputation* of a theorem prover or proof checker in which we place our trust. Unfortunately, reputation has limitations.

Reproducibility instead of reputation

Sir Francis Bacon's introduction of the scientific method—with its focus on reproducible results—was seen by the academics at that time as a way out of the political and social chaos that arose from the English Civil War (1642-1651).

Bacon's thoughts were enshrined in the Royal Society's creed "Nullius in verba" (take no one's word for it): that is, before trusting something, check it for yourself.

Who checks the proof checkers?

A familiar and ancient conundrum is “Quis custodiet ipsos custodes?” (Who will guard the guards?).

There is a modern approach to solving this problem: make it possible for anyone and everyone to monitor and audit the guards (proof checkers, in our case).

In 50 years, skeptics should be able to write their own checkers in order to re-check a formal proof.

Thus the format and semantics of documents containing formal proofs must be neither proprietary nor technology-specific: this is possible if the format has a well defined mathematical semantics.

State of the art: WWW

Digital signatures determine authorship of signed information, but few techniques are available to provide trust in what is actually claimed.

Blockchains and Merkle trees can help establish provenance and dependency.

The web has changed from a *cooperative* to an *adversarial* environment. Formal proofs can provide winning strategies against bad guys.

The WWW provides *frameworks* on which new behaviors *emerge*.

State of the art: Formal proof

Almost all formal proofs are dependent on the specific piece of technology that created them.

If the version number of the prover changes, proofs may not longer be proofs.

This situation is slowly improving.

- Specialized proof certificates: DRAT/DRUP, CPF, primality certificates, etc
- Proof-Carrying Code (PCC) [Necula & Lee, 1996]
- Frameworks for logics and proofs: logical frameworks (LF), mathematical knowledge management
- General-purpose proof certificates: Dedukti (Dowek, et al.), Foundational Proof Certificates (FPC) (Miller, et al.)

One can now imagine a proposal such as the following for providing formal proofs as an overlay on the WWW.

The world wide web of documents and proofs

	WWW of documents	**WWW of proof**

Standards and Infrastructure

Documents	Files in various formats	Proofs in various formats
Standards	SGML, HTML, etc	FPC, Dedukti, CPF, RUP, etc
Naming	URI, DOI	Content addressable storage
Transport	HTTP, FTP, torrents	In addition: IPFS
Trust	certificate authorities, public logs, encryption, open source, etc	*Reputation* (eg, proved by Coq 8.1) & *Reproducibility* (rechecking proof evidence)

Emergent structures

Access	browsers, JavaScript	interacting with proofs, proof browsers
Curation	Wikipedia, etc	proof libraries, textbooks

Not discussed more here.

The starting point: [Principle says Assertion]

It seems that we are forced to deal with a logic similar to "logics for access control" (e.g., Abadi, Burrows, Lampson, and Plotkin 1993).

P says String $\qquad$ P says $(\vdash B)$ $\qquad$ P says $(\Xi \vdash B)$

The truth of $[P\ \textit{says}\ A]$ is given by a cryptographic signing using the private key of P of (the string/file denoting) A.

If [Coq8.1 says $(\vdash B)$] and [HOL6.5 says $(\vdash B)$] then [I say $(\vdash B)$].

It is more likely that in 50 years, it will be proof certificates that are rechecked multiply ways.

If Ker12 says $(\Xi \vdash B)$ and Check51 says $(\Xi \vdash B)$ then I say $(\vdash B)$.

A logic for tracking trust seems necessary

P speaks-for Q: $\forall A.(P \text{ says } A \supset Q \text{ says } A)$

SAT1.5 speaks-for Vampire   SMT2.2 speaks-for Vampire

Thus, the two proof engines SAT1.5 and SMT2.2 are trusted internally to Vampire without checking them.

We need a logic to help track “trust-chains”.

Issues such as “degree of trust” or “transitivity of trust” are not forced on us yet.

A first attempt at a worthy goal

> *Goal: Construct a large library of formalized mathematics using Mizar, Coq, Agda, Isabelle, etc.*

This is a commonly stated problem: one articulation of it was the QED manifesto (1994).

The most important reasons offered by Freek Wiedijk for why the QED effort failed to advance

> *"is that only very few people are working on formalization of mathematics." (2007)*

There is no compelling application for fully mechanized mathematics among "working mathematicians", the intended target of QED. (An early suggested topic for the QED project was ring theory.)

The audience is too small.

Take a lesson from the early web

Tim Berners-Lee set out to build a repository of physics and engineering documents for CERN. What if he succeeded in doing (only) that?

The web could have taken another decade or so to emerge from that Ivory Tower prison.

Can we move trust and formal proof from the Ivory Tower?

Let's try for something much more ambitious

Goal 1: Libraries of formalized mathematics.

Goal 2: Reproducibility in science.
Formal proofs can capture the collection of data values, computation on them, statistical inference, etc.

Goal 3: Security and correctness of mobile and modular computing platforms.
"Is this app safe to put on my mobile phone?"
(Echos of Proof Carrying Code).

Goal 4: Journalism and "fake news".
While unlikely to use rich logic and proof techniques, journalism could benefit from the infrastructure of transparency and the signing of assertions that accompanies the infrastructure of proofs.

Some specific challenges for this project

Challenge 1: Permanent, signed electronic documents
Cryptographic hash functions and content-addressable storage (CAS) (e.g., BitTorrent and the Interplanetary File System IPFS) can be used to provide the permanence of such signed documents.

Challenge 2: Structuring libraries of theorems and proofs
Bindings (such as eigenvariables) can be implemented locally via, say, de Bruijn numerals and distributively via nonces.

Challenge 3: Interoperatbility of proofs
Provide tools for moving between implicit and explicit proofs. A proof using a naive approach to foundations might be moved to different formalizations of foundations.

Challenge 4: Replication in experimental sciences
Link traditional tools (Maple, Sage) and new web-based services (Open Science Framework (`osf.io`), Life Sciences Protocol Repository (`www.protocols.io`) to proof certificates that include computation and inference.

Challenge 1: Permanent, signed electronic documents
Challenge 2: Structuring libraries of theorems and proofs

The main principle:

> *Mathematical theorems and proofs should be permanent and global. Their structure and meaning must be defined using universal techniques.*

A consequence: they should not exist as some file in a directory structure located at some `URL` determined by the proof assistant's GitHub repository.

Well established alternatives exist.

- CAS - Content addressed storage. The hash is the name.
- IPFS - Interplanetary file system.
- PKI - Public key infrastructure

An example

```
Kind nat    type.
Type z      nat.
Type s      nat -> nat.

Define nat : nat -> prop by
 nat z ;
 nat (s N) := nat N.
```

```
> sha256sum nat.thm
67c1714291237a037d2f1d17ea1d662eaa32734bbeaa659ea05549c3266e76e0  nat.thm
>
```

```
import "67c1714291237a037d2f1d17ea1d662eaa32734bbeaa659ea05549c3266e76e0"

Define plus : nat -> nat -> nat -> prop by
  plus z N N ;
  plus (s N) M (s P) := plus N M P.

Theorem plus_comm : forall M N K, nat K -> plus M N K -> plus N M K.
  induction on 2. intros. case H2. apply plus_zero to H1. search.
    case H1. apply IH to H4 H3. apply plus_succ to H5. search.
```

How universal are logics and proofs?

For logical formulas, a good candidate is Church's simple theory of types [1940].

- Captures propositional, first-order, higher-order.
- Supports the addition of modal operators, fixed points, etc.

For inference rules and proofs, a good candidate is Gentzen's sequent calculus [1935] with modern enhancements (focusing [1992]).

- Captures classical and intuitionistic logic and arithmetic.
- Can encode also natural deduction, Frege proofs, etc.

Universal approaches to specifying syntactic structures

- Finite state machines / regular languages - used to tokenize string input
- Context-free grammars / attribute grammars - used to parse token lists into abstract syntax
- Foundational proof certificates - used to define the *inferential semantics* in proof evidence

A common thread with all three of these specification devices is that they are *declarative*. Logic programming provides a natural way to turn such specifications into executable parsers and proof checkers.

Conclusion

By realizing that proof checking must be reproducible, we are lead to consider an infrastructure in which agents declare what they assert and for other agents to explicitly trust or not such assertions.

If done correctly, formal proofs can provide a high degree of trustworthiness since they can enable rechecking by third parties.

This same infrastructure should be usable also in areas such as journalism and reproducibility in science.

Questions?