About Trust and Proof:
An experimental and heterogeneous framework

Dale Miller

Inria Saclay &
LIX, École Polytechnique
Partout Team

Joint work with
Farah Al Wardani &
Kaustuv Chaudhuri

Dagstuhl, 19 August 2024

Art by Nadia Miller

Outline

Trust in the digital world

The community of proof checkers

Distributed Assertion Management Framework (DAMF)

Benefits of a move towards DAMF

Outline

Problematic information sources in the internet age

Propaganda and misinformation

- A government asserts "The average life expectancy is 72 years and increasing" although the real value is 68 years and decreasing.

Disinformation campaigns

- "Firehose of Falsehood" Propaganda Model
- Steve Bannon: flood the zone with disinformation.
- The goal is disorientation and not persuasion.

Perverse Financial incentives

- Supporting grievances brings in clicks and revenue.

The digital world has greatly enabled disinformation and propaganda.

What about cryptographically signing all assertions

Journalists write documents that cite other documents, photos, spreadsheets, etc, all of which can be signed by others.

Journalists and editors publish documents signed either individually or collectively.

Consumers would then have allow-lists of agents they are willing to trust.

This sounds terribly naive.

Naive ideas sometimes need to be explored

Some naive solutions to important problems:

- Worried that people are bringing guns on planes and hijacking the planes? (1960s) Introduce metal detectors and screen everyone (1973).
- Worried that your mobile phone is giving out too much information about you and your location? Have your phone lie for you. Differential privacy.
- Worried that the binary file you are downloading could be a security risk on your computer? Have it paired with a proof that it is not dangerous. Proof-carrying code.
- Worried that the documents you are getting are forged, fake, or generated by an internet bot farm? *Have all documents cryptographically signed by their authors. Is this a solution?*

Naive ideas can also be terribly wrong

Archie Bunker (1972, All in the family) had a naive solution to hijacking.

- "All you gotta do is arm all your passengers, then your airlines, then they wouldn't have to search the passengers on the ground no more. They just pass out the pistols at the beginning of the trip, and they pick 'em up again at the end."

A shift in scope

We do not have expertise to address this "crisis in journalism."

The sign-everything-by-trusted-parties approach is used in some computer systems.

- boot loading: Secure Boot, UEFI, etc.
- software updates: Debian Secure apt, etc.

Some of the problems surrounding trust in the digital world reappear in the world of a mechanized proof-checking systems (where we have more expertise).

My focus today:

- *How can trust be implemented within the theorem proving community?*
- *Explore the costs and benefits of a particular approach*

Outline

Proof checking has a long history

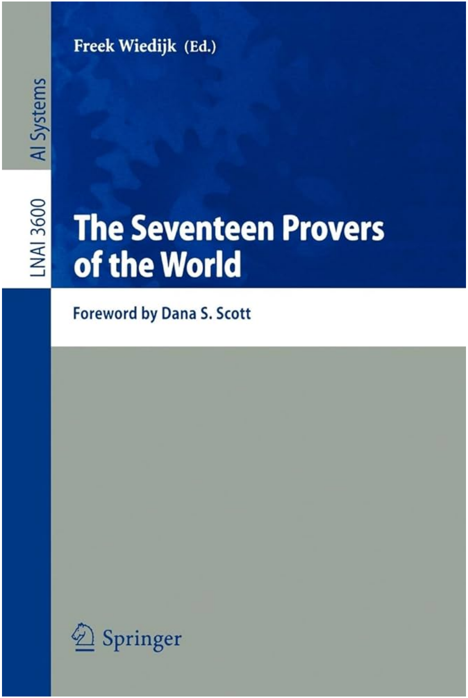
Leibniz's "universal symbolic language" and calculemus: "Let us calculate." (c. 1666).

Gordon, Milner, & Wadsworth, "Edinburgh LCF: A Mechanised Logic of Computation", 1979.

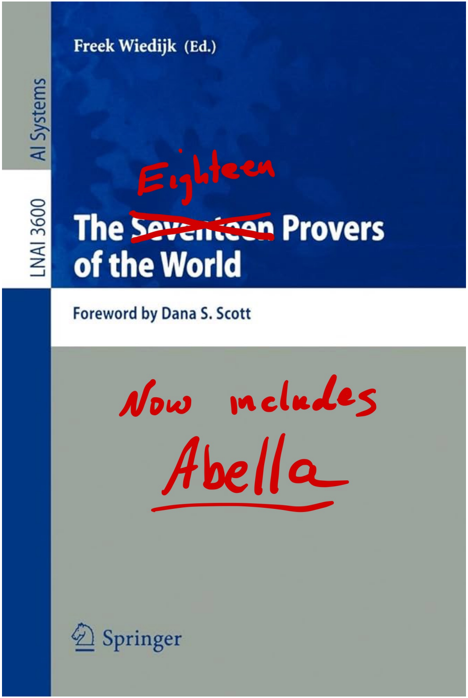
- The ML programming language (precursor to OCaml) was first designed as the meta-language for building a proof checker.

Many ambitious provers have been built since LCF.

- Boyer-Moore (1979), Isabelle (1989), Coq (1989), HOL (1988), PVS (1992), Lean (2013), ...



Published in 2006.



Abella appeared in 2009.

Trust no one else or trust a few

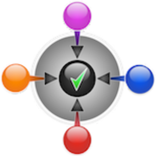
Most interactive theorem provers are *autarkic*: they only trust their own proof checking kernels.

- HOL, Isabelle, Coq, Lean

Some deductive program verification systems explicitly exploit and trust other theorem provers.

- Why3 - relies on external theorem provers to discharge verification conditions: CVC4, Z3, Coq, etc.
- TLA+ Proof System (TLAPS) - relies on back-end provers such as Isabelle, Zenon, and SMT solvers CVC3, Yices, Z3.

Dedukti and support for autarkic systems



The original goal of Dedukti (Dowek, et al., 2016)

- Get many provers (e.g., HOL, Isabelle, Coq) to output their proofs into the clean, simple format provided by Dedukti.
- Proof checkers for Dedukti are so simple, anyone can write one. Reference checkers exist too.
- Such independent proof checking instills more confidence.

A new goal of Dedukti

- If Coq needs a proof from Isabelle, then Isabelle exports it to Dedukti, and Dedukti outputs it for Coq.
- Dedukti is now a new tool for autarkic provers.

Another approach to managing proof and trust on the web

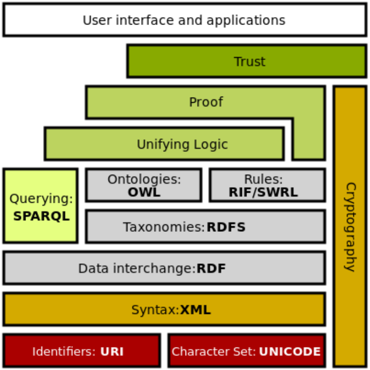
Semantic Web Stack

From Wikipedia, the free encyclopedia

The **Semantic Web Stack**, also known as **Semantic Web Cake** or **Semantic Web Layer Cake**, illustrates the architecture of the Semantic Web.

The Semantic Web is a collaborative movement led by international standards body the World Wide Web Consortium (W3C).[1] The standard promotes common data formats on the World Wide Web. By encouraging the inclusion of semantic content in web pages, the Semantic Web aims at converting the current web, dominated by unstructured and semi-structured documents into a "web of data". The Semantic Web stack builds on the W3C's Resource Description Framework (RDF).[2]

Contents [hide]



“Trust requires proof” vs “Proof requires trust”

We are familiar with the mathematician’s perspective that “Trust requires proof”.

Extending that argument beyond mathematics (to politics, journalism, etc) might have a future, but we are exploring the converse here.

Formal proofs can only be checked by computer programs.

Computer programs can be wrong.

Indeed, carefully designed and constructed proof checkers have been found to have errors (i.e., proofs of false).

We must speak explicitly about trusting proof checkers.

Outline

DAMF uses off-the-shelve technology: PKI and JSON

- Public-key infrastructure: private/public keys for cryptography
- JSON: an open standard approach to serialization of structured data: lists of attribute-values pairs

An example of a DAMF object.

```
{ "format":    "assertion",
  "agent":     "-----BEGIN PUBLIC KEY-----\nMFIwEA....",
  "signature": "3040021e10db76a6606d7a813747849028c79e...."
  "claim":     {"/": "bafyreibvtxzqhvht5rfxpw3rkgx...." },
}
```

DAMF uses off-the-shelve technology: IPFS

InterPlanetary File System

All files—local or remote—are referenced through a unique identifier which is a hash called a CID (content identifier).

```
> ipfs add nats.thm
added Qmf44WNArKxLC2MYmzK6DTAmkSSwAQwqvRmn94NRbStKxD nats.thm

> ipfs cat Qmf44WNArKxLC2MYmzK6DTAmkSSwAQwqvRmn94NRbStKxD
Kind nat type.
Type z nat.
Type s nat -> nat.

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

`ipfs://Qmf44WNArKxLC2MYmzK6DTAmkSSwAQwqvRmn94NRbStKxD`

DAMF uses off-the-shelve technology: IPLD

IPLD: Interplanetary Linked Data.

This provides an elegant integration of JSON with IPFS.

If `CID` points to a JSON object with attribute `"formula:"` then `CID/formula` is the CID for the attribute's value.

DAMF structures

Different formats used within DAMF. These are all values in IPFS.

Languages	naming a language (indicates how to parse)
Tools	names of theorem provers with their version info, etc. e..g., `Coq 8.16.1`, `Abella 2.0.9`, etc.
Contexts	typing declarations, definitions, etc
Formulas	Logical formulas
Sequents	list of dependencies with conclusion
Assertions	an agent signs a sequent

Others items needed: Productions / Collections / etc.

The Dispatch tool

Dispatch is an intermediary tool for publishing, retrieving, and analyzing trust in DAMF.

Dispatch specifies a family of JSON-based formats for DAMF objects and implements the main DAMF processes.

- Production of DAMF objects
- Consumption of DAMF objects
- Lookup: analyze dependencies to see who I am trusting

Dispatch can be used by both human users and provers.

Dispatch removes the need for a theorem proving system to be aware of IPFS.

A DAMF-aware prover only needs to be able to build and parse certain JSON objects from its theory files.

An agent makes an assertion: K *says* $(\Gamma \vdash B)$

B is the proposed theorem.

Γ lists the dependencies.

K is a pair of an agent (via a public key) and a mode.

A mode can be:

- `axiom` - used seldom: no proof is expected
- `conjecture` - an agent declares an interest in having this proved
- `tool T` - the agent used prover `T`
- null - the agent takes full responsibility

Note that a prover does not do the signing. The operator of the prover does the signing: binaries can be corrupted by users.

The truth of K *says* $(\Gamma \vdash B)$ comes from checking the digital signature.

The says logic: some inference

We permit just two inference rules for reasoning in the says logic.

$$\frac{K \textit{ says } (\Gamma_1 \vdash M) \qquad K \textit{ says } (M, \Gamma_2 \vdash N)}{K \textit{ says } (\Gamma_1, \Gamma_2 \vdash N)} \textsc{ Compose}$$

Assume that that K_1 is in the user-specified <u>allow list</u> of K_2.

Thus, K_1 <u>speaks for</u> K_2, which we write as $[K_1 \mapsto K_2]$.

$$\frac{K_1 \textit{ says } S}{K_2 \textit{ says } S} \textsc{ Trust}[K_1 \mapsto K_2]$$

The says logic is weak by design

There are many variations to access control logic in the literature, where the following rules might be assumed.

$$\frac{\Gamma \vdash N}{K \textit{ says } (\Gamma \vdash N)} \qquad \text{or} \qquad \frac{K \textit{ says } (\Gamma \vdash N)}{K \textit{ says } (K \textit{ says } (\Gamma \vdash N))}.$$

Such rules are *neither syntactically well-formed nor desirable* here.

No logical closure is assumed: let N_A, $N_{A \to B}$, and N_B be the formula objects that correspond to the formulas A, $A \to B$, and B.

We *do not* assume that the following rule is admissible:

$$\frac{K \textit{ says } (\Gamma \vdash N_{A \to B}) \qquad K \textit{ says } (\Gamma \vdash N_A)}{K \textit{ says } (\Gamma \vdash N_B)}\ \mathrm{MP}.$$

An interactive theorem prover well-suited for reasoning about the meta-theory of languages and logics involving binding.

- Various results on the λ-calculus involving big-step evaluation, small-step evaluation, and typing judgments
- Cut-admissibility for a sequent calculus
- Part 1a and Part 2a of the POPLmark challenge
- Some π-calculus meta-theory
- Takahashi's proof of the Church-Rosser theorem
- Tait's logical relations proof of weak normalization for STLC
- Girard's proof of strong normalization of STLC

Abella: A System for Reasoning about Relational Specifications by Baelde, Chaudhuri, Gacek, Miller, Nadathur, Tiu, and Wang. J. of Formalized Reasoning 7(2), 2014, 1-89.

An example of using DAMF

Let $\texttt{fib}(n)$ denotes the n^{th} Fibonacci number.

We want to have the following theorem available in Abella.

For $n \in \mathbb{N}$*,* $\texttt{fib}(n) = n^2$ *if and only if* $n \in \{0, 1, 12\}$*.*

We build the proof using three provers as follows.

1. We use λProlog to compute $\texttt{fib}(n)$ and n^2 for $n \in \{0, 1, \ldots, 12\}$ and to compare them for equality.
2. We use Coq to prove: forall n, if $n \geq 13$ then $\texttt{fib}(n) > n^2$.
3. We use Abella to do all the remaining steps.

These systems use different syntaxes, logics, and proofs.

This integration was achieved with minor additions/modifications to printers and parsers: no kernels were touched.

https://distributed-assertions.github.io/example-walkthrough/

Daily Dale Pools Libs Inria Bard deepl All Bookmarks

Distributed Assertion Management Framework

Search

Example Walkthrough

Back to top

This walkthrough shows how to use a combination of DAMF-aware edge systems to verify and subsequently publish the following assertion:

Theorem. For $n \in \mathbb{N}$, $\mathrm{fib}(n) = n^2$ if and only if $n \in \{0, 1, 12\}$, where $\mathrm{fib}(n)$ stands for the nth Fibonacci number defined as: $\mathrm{fib}(0) \triangleq 0$, $\mathrm{fib}(1) \triangleq 1$, and $\mathrm{fib}(n+2) \triangleq \mathrm{fib}(n+1) + \mathrm{fib}(n)$.

In the *if* direction, the assertion is fairly easy to prove in any system that can support inductive definitions such as fib in the first place: one just has to compute $\mathrm{fib}(0)$, $\mathrm{fib}(1)$, and $\mathrm{fib}(12)$ and verify that they are indeed 0, 1, and 144, respectively. The *only if* direction, on the other hand, requires the ability to reason about the growth of the Fibonacci function with respect to the quadratic function $n \mapsto n^2$. In particular, one needs the following lemma:

Lemma. For $n \in \mathbb{N}$, if $n \geq 13$, then $\mathrm{fib}(n) > n^2$.

Table of contents

Multilanguage situations

Every formula object packages the formula with its context and language identifier.

Thus, every formula object is independent of every other formula object.

In a sequent $N_1 \vdash N_0$, there is no requirement that the conclusion N_0 and the dependency N_1 be in the same language or have a common context.

In the autarkic setting, sequents will generally use the same language and context for all formula objects.

In the wider non-autarkic world, we can use multilingual sequents.

Multilingual sequents

For a theorem written in the one prover to be used by a different prover, we need to transform a formula object in the first language to a corresponding object in the second language.

`Coq 8.16.1`:

```
Theorem ex_coq :
  forall n:nat, 8 <= n -> lincomb n 3 5.
```

`Abella 2.0.9`:

```
Import "nats".
Define lincomb : nat -> nat -> nat -> prop by
  lincomb N J K := exists X Y U V,
    times X J U /\ times Y K V /\ plus U V N.
Theorem ex_ab :
   forall n, nat n -> le 8 n -> lincomb n 3 5.
```

Such a translation is often sophisticated: here, the function symbols $+$ and $*$ are replaced by relations in Abella.

Language adapters

Adapters are tools that do such translations.

The sequent that represents this translation has the form

$$\langle \texttt{Coq 8.16.1}, \Sigma_{\texttt{ex\_coq}}, \texttt{ex\_coq} \rangle \vdash \langle \texttt{Abella 2.0.9}, \Sigma_{\texttt{ex\_ab}}, \texttt{ex\_ab} \rangle.$$

Suppose agent K_1 signs this translation and that agent K_2 signs the sequent $\vdash \langle \texttt{Coq 8.16.1}, \Sigma_{\texttt{ex\_coq}}, \texttt{ex\_coq} \rangle$.

If K_1 and K_2 are trusted by the user of `Abella 2.0.9`, then the formula object $\langle \texttt{Abella 2.0.9}, \Sigma_{\texttt{ex\_ab}}, \texttt{ex\_ab} \rangle$ can be treated as a theorem by that user.

Outline

Trust in the digital world

The community of proof checkers

Distributed Assertion Management Framework (DAMF)

Benefits of a move towards DAMF

Cost and Benefits

The potential *costs* seem clear.

- New standards and processes need to be adopted.
- Another layer of software is needed (IPFS, Dispatch, etc).
- Agents need to sign theorems.

Some potential *benefits* for the community.

- Many existing tools can be used as adapters (e.g., Dedukti).
- Non-incumbent theorem provers (such as Abella) can play their niche role in larger projects.
- Specialized proof languages for specialized settings can be built as adapters: e.g., graphic presentations of commuting diagrams.
- No explicit library structure is imposed. Hierarchies of theories can become emergent structures.
- When adversaries show up, we can keep them out of our allow-lists.

Benefits for an individual theorem prover

- Version control tracking.
- Since the emphasis is not on rechecking, the performance of kernels can be relaxed; efforts to optimize kernels often introduce bugs.
- Features can be added without touching the kernel: e.g.: polymorphism, higher-order features.
- Web-centric theorem provers: IPFS provides a solution to file persistence.

abella-prover.org/tutorial/try/

Daily Dale Pools Libs Inria Bard deepl All Bookmarks

Load 4.1 - Automata or start from scratch

Specification Reasoning

Use `Ctrl-Return` (mac: `Cmd-Return`) to process up to cursor.

```
Kind st,lab       type.
Type p0,p1,q0,q1  st.
Type a,b          lab.

Define step : st -> lab -> st -> prop by
  step p0 a p0 ; step p0 b p1 ;
  step q0 a q1 ; step q1 a q0.

CoDefine sim : st -> st -> prop by
  sim P Q :=
    forall L Pn, step P L Pn ->
      exists Qn, step Q L Qn /\ sim Pn Qn.

Theorem q0_sim_p0 : sim q0 p0.
coinduction. unfold. intros.
case H1. witness p0. split. search.
unfold. intros.
case H2. witness p0. split. search.
search.
```

```
q0_sim_p0 < unfold.

CH : sim q0 p0 +
============================
 forall L Pn, step q0 L Pn -> (exists (

q0_sim_p0 < intros.

Variables: L Pn
CH : sim q0 p0 +
H1 : step q0 L Pn
============================
 exists Qn, step p0 L Qn /\ sim Pn Qn +

q0_sim_p0 < case H1.

CH : sim q0 p0 +
============================
 exists Qn, step p0 a Qn /\ sim q1 Qn +

q0_sim_p0 < witness p0.

CH : sim q0 p0 +
============================
```

Process Full Reset

Made with the help of Ace, AngularJS, Bootstrap, jQuery, `js_of_ocaml`, and OCaml. © 2015, Inria

Conclusions

- Trusting a formal proof requires trusting a computer program.
- DAMF attempts to explicitly address this notion of trust by
  - having users sign their assertions and
  - maintaining all dependencies in a global file system.
- Using the Dispatch tool, a prover can be DAMF-aware with minimal modifications to the printing and parsing subsystems of a prover.
- It is still open if others in the community find DAMF appealing.
- From the perspective of Abella, we see many benefits of using DAMF.

Thanks

Questions?

Art by Nadia Miller