Foundational Proof Certificates

Making proof universal and permanent

Dale Miller

INRIA-Saclay & LIX, École Polytechnique

23 September 2013

Can we standardize, communicate, and trust formal proofs?

Joint work with Zakaria Chihani and Fabien Renaud.
Funded by the ERC Advanced Grant ProofCert.

Communicating and Checking Formal proofs

Our focus:

Computer agents communicating and checking formal proofs.

A *formal proof* is a document with a precise syntax that is machine generated and machine checkable.

We do not assume that formal proofs are human-readable.

Trusted computer tools are used to check proofs so that humans come to trust the truth of a formula.

Provers: computer agents that produce proofs

There is a wide range of provers.

- automated and interactive theorem provers
- computer algebra systems
- model checkers, SAT solvers
- type inference, static analysis
- testers

There is a wide range of "proof evidence."

- proof scripts: steer a theorem prover to a proof
- resolution refutations, natural deduction, tableaux, etc
- winning strategies, simulations

Separate proofs from provenance

Most formal proofs are tied to some specific technology: change the version number and a proof script does not check anymore.

A bridge between two provers can be doubly fragile.

There are many advantages if provers publish their proofs as independently checkable objects:

- libraries, marketplaces, cooperation, etc.

We shall use the term "proof certificate" for those documents denoting proofs that are circulated between provers and checkers.

Four desiderata for proof certificates

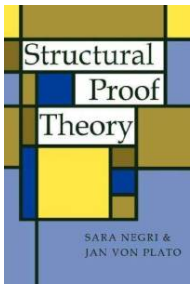
D1: A simple checker can, in principle, check if a proof certificate denotes a proof.

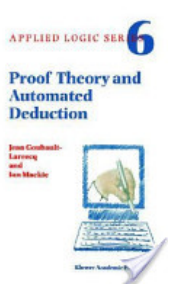
D2: The proof certificate format supports a broad spectrum of proof systems.

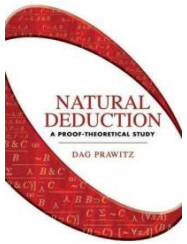
These two desiderata enable the creation of both **marketplaces** and **libraries** of proofs.

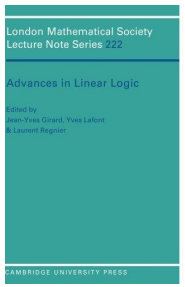
D3: A proof certificate is intended to denote a proof in the sense of structural proof theory.

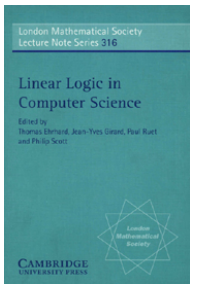
Structural proof theory is a mature field that deals with deep aspects of proofs and their properties.









For example: given certificates for

$$\forall x(A(x) \supset \exists y\ B(x, y)) \quad \text{and} \quad A(10),$$

can we extract from them a witness t such that $B(10, t)$ holds?

D4: A proof certificate can simply leave out details of the intended proof.

Formal proofs are often huge. All means to reduce their size need to be available.

- Allow abstractions and lemma.
- Separate *computation* from *deduction* and leave computation traces out of the certificate.
- Permit holes in proofs: we now have a trade-offs between *proof size* and *proof reconstruction* via (bounded) proof search.

Proof checking may involve significant computation in order to *reconstruct* missing proof details.

Which logic?

First-order or higher-order?

Which logic?

First-order or higher-order? Both!

Higher-order (à la Church 1940) seems a good choice since it includes propositional and first-order.

Which logic?

First-order or higher-order? Both!

Higher-order (à la Church 1940) seems a good choice since it includes propositional and first-order.

Classical or intuitionistic logic?

Which logic?

First-order or higher-order? Both!

Higher-order (à la Church 1940) seems a good choice since it includes propositional and first-order.

Classical or intuitionistic logic? Both!

Imagine that these two logics fit together in one larger logic. Following Gentzen (LK/LJ), Girard (LU), Liang & M (LKU, PIL).

Which logic?

First-order or higher-order? Both!

Higher-order (à la Church 1940) seems a good choice since it includes propositional and first-order.

Classical or intuitionistic logic? Both!

Imagine that these two logics fit together in one larger logic. Following Gentzen (LK/LJ), Girard (LU), Liang & M (LKU, PIL).

Modal, temporal, spatial?

Many modal logics are adequately encoded into first-order logic ... but there is likely to always be a frontier that does not fit well.

Earliest notion of formal proof

Frege, Hilbert, Church, Gödel, etc, made extensive use of the following notion of proof:

> *A proof is a list of formulas, each one of which is either an* axiom *or the conclusion of an* inference rule *whose premises come earlier in the list.*

While granting us trust, there is little useful structure here.

The first programmable proof checker

LCF/ML (1979) viewed proofs as slight generalizations of such lists.

ML provided types, abstract datatypes, and higher-order programming in order to increase confidence in proof checking.

Many provers today (HOL, Coq, Isabelle) are built on LCF.

Atoms and molecules of inference

Atoms of inference

- Gentzen's **sequent calculus** first provided these: introduction, identity, and structural rules.
- Girard's **linear logic** refined our understanding of these further.
- To account for first-order structure, we also need **fixed points** and **equality**.

Rules of Chemistry

- Focused proof systems show us that certain pairs of atoms stick together while others pairs form boundaries.

Molecules of inference

- Collections of atomic inference rules that stick together form synthetic inference rules (molecules of inference).

Satisfying the desiderata

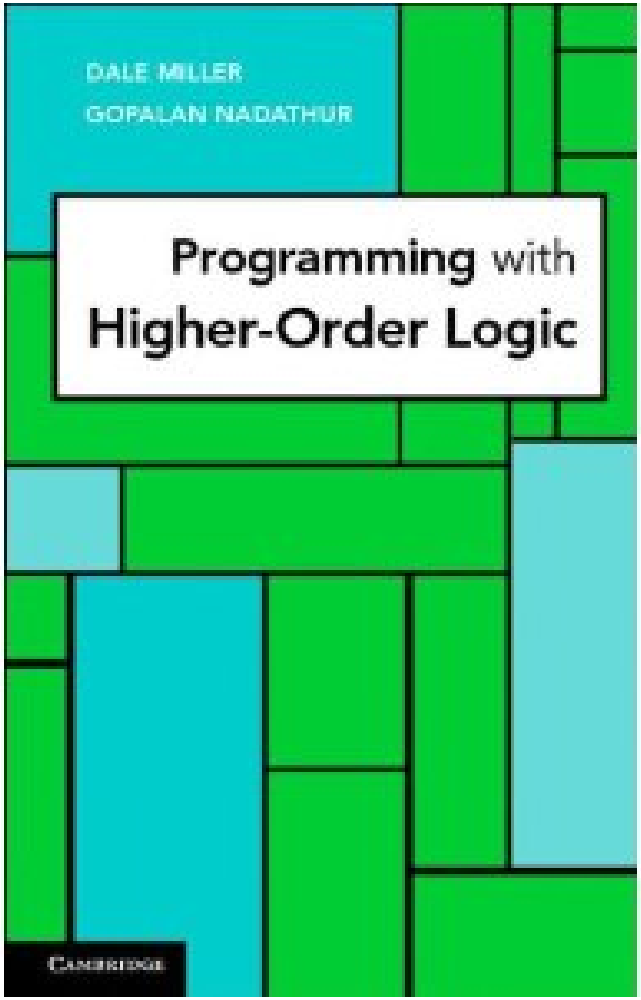
D1: Simple checkers.
Only the atoms of inference and the rules of chemistry (both small and closed sets) need to be implemented in the checker.

D2: Certificates supports a wide range of proof systems.
The molecules of inference can be engineered into a wide range of existing inference rules.

D3: Certificates are based on proof theory.
Immediate by design.

D4: Details can be elided.
Search using atoms will match search in the space of molecules, ie., don't invent new molecules in the checker.

Safe proof reconstruction via logic programming



Logic programming can check proofs in sequent calculus.

Proof reconstruction requires unification and (bounded) proof search.

The λProlog programming language [M & Nadathur, 1986, 2012] also include types, abstract datatypes, and higher-order programming.

An analogy between SOS and FPC

Structural Operational Semantics

1. There are many programming languages.

An analogy between SOS and FPC

Structural Operational Semantics

1. There are many programming languages.
2. SOS can define the semantics of many of them.

An analogy between SOS and FPC

Structural Operational Semantics

1. There are many programming languages.
2. SOS can define the semantics of many of them.
3. Logic programming can provide prototype interpreters.

An analogy between SOS and FPC

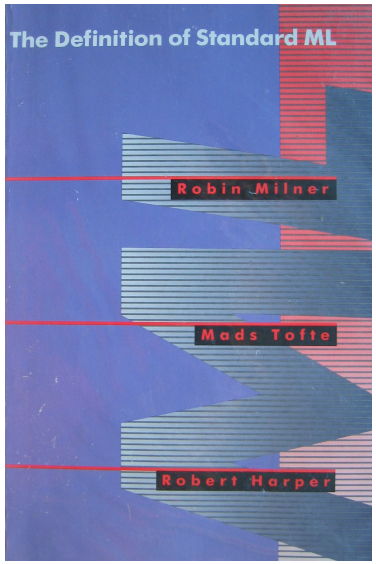
Structural Operational Semantics

1. There are many programming languages.
2. SOS can define the semantics of many of them.
3. Logic programming can provide prototype interpreters.
4. Compliant compilers can be built based on the semantics.

An analogy between SOS and FPC

Structural Operational Semantics

1. There are many programming languages.
2. SOS can define the semantics of many of them.
3. Logic programming can provide prototype interpreters.
4. Compliant compilers can be built based on the semantics.



An analogy between SOS and FPC

Structural Operational Semantics

1. There are many programming languages.
2. SOS can define the semantics of many of them.
3. Logic programming can provide prototype interpreters.
4. Compliant compilers can be built based on the semantics.

Foundational Proof Certificates

1. There are many forms of proof evidence.
2. FPC can define the semantics of many of them.
3. Logic programming can provide prototype checkers.
4. Compliant checkers can be built based on the semantics.

Synchronous/positive/non-invertible rules and their experts

$$\frac{}{\vdash \Theta \Downarrow t^+} \qquad \frac{\vdash \Theta \Downarrow B_1 \qquad \vdash \Theta \Downarrow B_2}{\vdash \Theta \Downarrow B_1 \wedge^+ B_2}$$

$$\frac{\vdash \Theta \Downarrow B_i \qquad i \in \{1, 2\}}{\vdash \Theta \Downarrow B_1 \vee^+ B_2}$$

$$\frac{\vdash \Theta \Downarrow [t/x]B}{\vdash \Theta \Downarrow \exists x.B}$$

Synchronous/positive/non-invertible rules and their experts

$$\frac{true_e(\Xi)}{\Xi \vdash \Theta \Downarrow t^+} \qquad \frac{\Xi_1 \vdash \Theta \Downarrow B_1 \quad \Xi_2 \vdash \Theta \Downarrow B_2 \quad \wedge_e(\Xi, \Xi_1, \Xi_2)}{\Xi \vdash \Theta \Downarrow B_1 \wedge^+ B_2}$$

$$\frac{\Xi' \vdash \Theta \Downarrow B_i \qquad i \in \{1, 2\} \qquad \vee_e(\Xi, \Xi', i)}{\Xi \vdash \Theta \Downarrow B_1 \vee^+ B_2}$$

$$\frac{\Xi' \vdash \Theta \Downarrow [t/x]B \qquad \exists_e(\Xi, \Xi', t)}{\Xi \vdash \Theta \Downarrow \exists x.B}$$

Asynchronous/negative/invertible rules and their clerks

$$\frac{\Xi' \vdash \Theta \Uparrow \Gamma \quad f_c(\Xi, \Xi')}{\Xi \vdash \Theta \Uparrow f^-, \Gamma} \qquad \frac{\Xi' \vdash \Theta \Uparrow A, B, \Gamma \quad \vee_c(\Xi, \Xi')}{\Xi \vdash \Theta \Uparrow A \vee^- B, \Gamma}$$

$$\frac{}{\Xi \vdash \Theta \Uparrow t^-, \Gamma} \qquad \frac{\Xi_1 \vdash \Theta \Uparrow A, \Gamma \quad \Xi_2 \vdash \Theta \Uparrow B, \Gamma \quad \wedge_c(\Xi, \Xi_1, \Xi_2)}{\Xi \vdash \Theta \Uparrow A \wedge^- B, \Gamma}$$

$$\frac{\Xi' \vdash \Theta \Uparrow [y/x]B, \Gamma \qquad \forall_c(\Xi, \Xi') \qquad y \text{ not free in } \Xi, \Theta, \Gamma, B}{\Xi \vdash \Theta \Uparrow \forall x.B, \Gamma}$$

The Structural Rules

$$\frac{\Xi' \vdash \Theta, \langle I, C\rangle \Uparrow \Gamma \quad \mathit{store}_c(\Xi, C, \Xi', I)}{\Xi \vdash \Theta \Uparrow C, \Gamma}\ \mathit{store}$$

$$\frac{\Xi_1 \vdash \Theta \Uparrow B \qquad \Xi_2 \vdash \Theta \Uparrow \neg B \qquad \mathit{cut}_e(\Xi, \Theta, \Xi_1, \Xi_2, B)}{\Xi \vdash \Theta \Uparrow \cdot}\ \mathit{cut}$$

$$\frac{\Xi' \vdash \Theta \Uparrow N \quad \mathit{release}_e(\Xi, \Xi')}{\Xi \vdash \Theta \Downarrow N}\ \mathit{release} \qquad \frac{\mathit{init}_e(\Xi, \Theta, I) \quad \langle I, \neg P_a\rangle \in \Theta}{\Xi \vdash \Theta \Downarrow P_a}\ \mathit{init}$$

$$\frac{\Xi' \vdash \Theta \Downarrow P \quad \mathit{decide}_e(\Xi, \Theta, \Xi', I) \quad \langle I, P\rangle \in \Theta \quad \text{positive}(P)}{\Xi \vdash \Theta \Uparrow \cdot}\ \mathit{decide}$$

Here, P is a positive formula; N a negative formula; P_a a positive literal; C a positive formula or negative literal.

Meta-theory of LKF (black bits on the previous 3 slides)

Let B be a formula and $\hat{B}$ (an annotation of B) result from B by placing $+$ or $-$ on t, f, $\wedge$, and $\vee$ (there are exponentially many such placements).

Theorem. If B is a classical theorem then every annotation $\hat{B}$ has an LKF proof. Conversely, if some annotation $\hat{B}$ has an LKF proof then B is a classical theorem. [Liang & M, TCS 2009].

- Different polarizations do not change *provability* but can radically change the *proofs*.
- Negative (non-atomic) formulas are treated linearly (never weakened nor contracted).
- Only positive formulas are contracted (in the Decide rule).

A similar proof system for intuitionistic logic exists (LJF) and the LKU proof system [Liang & M, 2011] unifies these (and MALLF) into one framework.

Inferences as logic program clauses

$$\forall\Theta\forall\Gamma.\ \mathtt{async}(\Theta, [t^{-}|\Gamma]).$$

$$\forall\Theta\forall\Gamma\forall A\forall B.\ \mathtt{async}(\Theta, [(A \wedge^{-} B)|\Gamma])\ \mathtt{:-}\ \mathtt{async}(\Theta, [A|\Gamma]),\ \mathtt{async}(\Theta, [B|\Gamma]).$$

$$\forall\Theta\forall\Gamma\forall A\forall B.\ \mathtt{sync}(\Theta, A \vee^{+} B)\ \mathtt{:-}\ \mathtt{sync}(\Theta, A);\ \mathtt{sync}(\Theta, B).$$

$$\forall\Theta\forall\Gamma\forall P.\ \mathtt{async}(\Theta, [])\ \mathtt{:-}\ \mathtt{memb}(P, \Theta),\ \mathtt{pos}(P),\ \mathtt{sync}(\Theta, P).$$

$$\forall\Theta\forall B\forall C.\ \mathtt{async}(\Theta, [])\ \mathtt{:-}\ \mathtt{negate}(B, C),\ \mathtt{async}(\Theta, B),\ \mathtt{async}(\Theta, C).$$

Example: Decision procedure using cnf

```
cnf :  cert
idx :  form -> index
```

$$f_c(\mathtt{cnf}, \mathtt{cnf}). \qquad \wedge_c(\mathtt{cnf}, \mathtt{cnf}, \mathtt{cnf}).$$

$$\forall C.\ \mathit{store}_c(\mathtt{cnf}, C, \mathtt{cnf}, \mathtt{idx}(C)). \qquad \vee_c(\mathtt{cnf}, \mathtt{cnf}).$$

$$\forall \Theta \forall I.\ \mathit{decide}_e(\mathtt{cnf}, \Theta, \mathtt{cnf}, I). \qquad \mathit{release}_e(\mathtt{cnf}, \mathtt{cnf}).$$

$$\forall \Theta \forall I \forall N.\ \mathit{init}_e(\mathtt{cnf}, \Theta, I).$$

Example: Resolution as a proof certificate

A *clause:* $\forall x_1 \ldots \forall x_n [L_1 \vee \cdots \vee L_m]$

- C_3 is a *resolution* of C_1 and C_2 if we chose the mgu of two complementary literals, one from each of C_1 and C_2, etc.

Let $\vdash_d \Theta \Uparrow \Gamma$ mean that $\vdash \Theta \Uparrow \Gamma$ has a proof with decide depth d.

Polarize using $\vee^-$ and $\wedge^+$.

- If C_3 is a resolvent of C_1 and C_2 then $\vdash_2 \neg C_1, \neg C_2 \Uparrow C_3$.

Example: Resolution as a proof certificate (cont)

Translate a refutation of $C_1, \ldots, C_n$ into an LKF proof with small holes as follows:

$$\dfrac{\Xi \qquad \dfrac{\vdots}{\vdash \neg C_1, \ldots, \neg C_n, \neg C_{n+1} \Uparrow \cdot}\, Store}{\vdash \neg C_1, \ldots, \neg C_n \Uparrow \neg C_{n+1}}\, Cut_p$$
$$\vdash \neg C_1, \neg C_2 \Uparrow C_{n+1}$$
$$\vdash \neg C_1, \ldots, \neg C_n \Uparrow \cdot$$

Here, Ξ can be replaced with a "hole" annotated with bound 2.

To capture more of the parallel structure present in a refutation dag, a "multicut" can be used here.

Example: Resolution

```
idx :  int -> index          dl :  list int -> cert
lit :  form -> index      ddone :  cert
```

$$\forall L.\ \vee_c(\mathtt{dl}(L), \mathtt{dl}(L)).$$
$$\forall L.\ f_c(\mathtt{dl}(L), \mathtt{dl}(L)).$$
$$\forall L.\ true_e(\mathtt{dl}(L))$$
$$\forall L.\ init_e(\mathtt{dl}(L), \Theta, I)$$
$$\forall C \forall L.\ store_c(\mathtt{dl}(L), C, \mathtt{dl}(L), (lit\ C)).$$
$$\forall L.\ release_e(\mathtt{dl}(L), \mathtt{dl}(L)).$$
$$\forall I \forall J.\ decide_e(\mathtt{dl}([I, J]), \Theta, \mathtt{dl}([J]), \mathtt{idx}(I)).$$
$$\forall I \forall J.\ decide_e(\mathtt{dl}([I, J]), \Theta, \mathtt{dl}([I]), \mathtt{idx}(J)).$$
$$\forall I.\ decide_e(\mathtt{dl}([I]), \Theta, \mathtt{dl}([]), \mathtt{idx}(I)).$$
$$\forall P.\ decide_e(\mathtt{dl}([]), \Theta, \mathtt{ddone}, \mathtt{lit}(P)).$$

$$\forall L.\ \forall_c(\mathtt{dl}(L), \mathtt{dl}(L)).$$
$$\forall L.\ \exists_e(T, \mathtt{dl}(L), \mathtt{dl}(L))$$
$$\forall \Theta \forall I.\ init_e(\mathtt{ddone}, \Theta, I)$$
$$\forall L.\ \wedge_e(\mathtt{dl}(L), \mathtt{dl}(L), \mathtt{dl}(L))$$

```
rdone :  cert
rlist :  list (int * int * int) -> cert
rlisti :  int -> list (int * int * int) -> cert
```

$$\forall I \forall \Theta.\ \mathit{decide}_e(\mathtt{rlist}([]), \Theta, \mathtt{rdone}, \mathtt{idx}(I))\ \mathtt{:-}\ \langle \mathtt{idx}(I), \mathit{true}\rangle \in \Theta.$$

$$\forall I, \ldots, \Theta. \mathit{cut}_e(\mathtt{rlist}([\langle I, J, K\rangle | R]), \Theta, \mathtt{dl}([I, J]), \mathtt{rlisti}(K, R), N)\ \mathtt{:-}\ \langle \mathtt{idx}(K), C\rangle \in \Theta, \mathtt{negate}(C, N).$$

$$\forall R.\ f_c(\mathtt{rlist}(R), \mathtt{rlist}(R)).$$

$$\forall C \forall I \forall R.\ \mathit{store}_c(\mathtt{rlisti}(I, R), C, \mathtt{rlist}(R), I).$$

$$\mathit{true}_e(\mathtt{rdone})$$

Adequacy of encoding

What are we really checking? Only soundness.

Consider resolution again. If $\vdash \neg C_1, \neg C_2 \Uparrow C_0$ is provable, C_0 may not be the resolvent of C_1 and C_2.

For example, the resolution of
$\forall x[p(x) \vee r(f(x))]$ and $\forall x[\neg p(f(x)) \vee q(x)]$ is
$\forall x[r(f(f(x))) \vee q(x)]$. But

$$\vdash \exists x[\neg p(x) \wedge \neg r(f(x))], \exists x[p(f(x)) \wedge \neg q(x)] \Uparrow r(f(f(f(a)))) \vee q(f(a)) \vee s(f(a)).$$

This formula is similar to a resolvent except it uses a unifier that is not most general and it has an additional literal.

What relations is there between LF and FPC?

We should be able to encode LF, LFSC (LF with side conditions) and LF modulo (Dedukti) as FPCs using focused LJF system.

Alone LF does not seem to have the right "atoms of inference."

- Canonical normal forms provide only one polarization: negative connectives and atoms (async on right, sync on left).
- They lack a notion of sharing and are redundant in their use of types.
- They lack a natural treatments of parallel proof steps.

Classical provability can be encoded via double-negation translations, but to mimic classical proofs with high fidelity, flexible polarizations are needed.

Future projects via ProofCert

Stage one

- LKU is a unifying framework for LKF, LJF, MALLF. Mechanically formalize its meta-theory.
- Treat typed λ-calculi fully: LF, LFSC, λP-modulo (Deduki)
- Design many more FPCs: Frege systems, linear reasoning, expansion trees, equality reasoning, DPLL, SAT, etc.
- Deployment. Competitions? TPTP?

Stage two

- Fixed points. Neither the proof theory nor unification (proof reconstruction) are well understood yet.

Stage three

- Incorporate partial proofs and counterexamples.