Defining the semantics of proof evidence

Dale Miller

INRIA-Saclay & LIX, École Polytechnique
Palaiseau, France

Joint work with Zakaria Chihani and Fabien Renaud, INRIA.
See: APPA 2014, CADE 2013, CPP 2011.

ITU, Copenhagen, 11 September 2014

The network *is* the prover

Sun Microsystems (1984): The network *is* the computer

The formal methods community uses many isolated provers technologies: proof assistants (Coq, Isabelle, HOL, PVS, etc), model checkers, SAT solvers, etc.

Goal: Permit the formal methods community to become a network of communicating provers.

We shall use the term "proof certificate" for those circulating documents denoting proofs.

Many computer systems producing many kinds of 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

If the necessary networking infrastructure is built, a wider range of provers and proof evidence would appear.

Separate proofs from provenance

Most formal proofs are tied to some specific technology: change a version number and a proof may no longer check.

We focus here on how we might separate proof from provenance.

- Provers *output* proof evidence for a theorem (via some "proof language").
- *Trusted* checkers must be available to check such evidence.

If we do our job right, proofs become a commodity and our attention turns other aspects of computer systems.

The need for frameworks

Three central questions:

- How can we manage so many “proof languages”?
- Will we need just as many proof checkers?
- How does this improve trust?

Computer scientists have seen this kind of problem before.

The need for frameworks

Three central questions:

- How can we manage so many "proof languages"?
- Will we need just as many proof checkers?
- How does this improve trust?

Computer scientists have seen this kind of problem before.

We develop frameworks to address such questions.

- lexical analysis: finite state machines / transducers
- language syntax: grammars, parsers, attribute grammars, parser generators
- programming languages: denotational and operational semantics

A framework for proof evidence: First pick the logic

Church's Simple Theory of Types (STT) is a good choice for the syntax of formulas.

Allow both classical or intuitionistic logic.

Propositional, first-order, and higher-order logics are easily identifiable sublogics of STT.

Many other logics can adequately be encoded into STT: eg, equational, modal, etc.

There is likely to always be a frontier of research that involves logics that do not fit well into a fixed framework. C'est la vie.

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.

More recent advances: 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).

Features enabled for proof certificates

- Simple checkers can be implemented.
  Only the atoms of inference and the rules of chemistry (both small and closed sets) need to be implemented in a checker of certificates.

- Certificates support a wide range of proof systems.
  The molecules of inference can be engineered into a wide range of inference rules.

- Certificates are based (ultimately) on proof theory.
  Immediate by design.

- Proof details can be elided.
  Search using atoms will match search in the space of molecules: that is, the checker will not invent new molecules.

An analogy between SOS and FPC

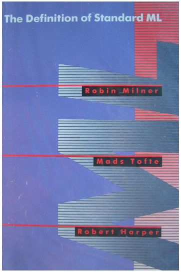
Structural Operational Semantics

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.

Clerks and experts: the office workflow analogy

Imagine an accounting office that needs to check if a certain mound of financial documents (provided by a **client**) represents a legal tax form (as judged by the **kernel**).

Experts look into the mound and extract information and

- *decide* which transactions to dig into and
- *release* their findings for storage and later reconsideration.

Clerks take information released by the experts and perform some computations on them, including their *indexing* and *storing*.

Focused proofs alternate between two phases: *synchronous* (experts are active) and *asynchronous* (clerks are active).

The terms *decide*, *store*, and *release* come from proof theory.

A proof certificate format defines workflow and the duties of the clerks and experts.

Proof checking and proof reconstruction

Clearly, (determinate) computation is built into this paradigm: the clerks can perform such computation.

Proof *reconstruction* might be needed when invoking not-so-expert experts.

Non-deterministic computation is part of the mix: non-determinism is an important resource that is useful for proof-compression.

The *LKneg* proof system

Use invertible rules where possible. In propositional classical logic, both conjunction and disjunction can be given invertible rules.

$$\frac{\vdash \cdot; B}{\vdash B}\ \mathit{start} \qquad \frac{\vdash \Delta, L; \Gamma}{\vdash \Delta; L, \Gamma}\ \mathit{store} \qquad \frac{}{\vdash \Delta, A, \neg A; \cdot}\ \mathit{init}$$

$$\frac{\vdash \Delta; \Gamma}{\vdash \Delta; \mathit{false}, \Gamma} \qquad \frac{\vdash \Delta; B, C, \Gamma}{\vdash \Delta; B \vee C, \Gamma} \qquad \frac{}{\vdash \Delta; \mathit{true}, \Gamma} \qquad \frac{\vdash \Delta; B, \Gamma \quad \vdash \Delta; C, \Gamma}{\vdash \Delta; B \wedge C, \Gamma}$$

Here, A is an atom, L a literal, Δ a multiset of literals, and Γ a list of formulas. Sequents have two *zones*.

This proof system is a decision procedure (resembling conjunctive normal forms). A small (constant sized) certificate is possible.

The *LKneg* proof system

Use invertible rules where possible. In propositional classical logic, both conjunction and disjunction can be given invertible rules.

$$\frac{\vdash \cdot; B}{\vdash B}\ \mathit{start} \qquad \frac{\vdash \Delta, L; \Gamma}{\vdash \Delta; L, \Gamma}\ \mathit{store} \qquad \frac{}{\vdash \Delta, A, \neg A; \cdot}\ \mathit{init}$$

$$\frac{\vdash \Delta; \Gamma}{\vdash \Delta; \mathit{false}, \Gamma} \qquad \frac{\vdash \Delta; B, C, \Gamma}{\vdash \Delta; B \vee C, \Gamma} \qquad \frac{}{\vdash \Delta; \mathit{true}, \Gamma} \qquad \frac{\vdash \Delta; B, \Gamma \quad \vdash \Delta; C, \Gamma}{\vdash \Delta; B \wedge C, \Gamma}$$

Here, A is an atom, L a literal, Δ a multiset of literals, and Γ a list of formulas. Sequents have two *zones*.

This proof system is a decision procedure (resembling conjunctive normal forms). A small (constant sized) certificate is possible.

Consider proving $(p \vee C) \vee \neg p$ for large C.

The *LKpos* proof system

Non-invertible rules are used here.

$$\frac{\vdash B; \cdot; B}{\vdash B}\ \mathit{start} \qquad \frac{\vdash B; \mathcal{N}, \neg A; B}{\vdash B; \mathcal{N}; \neg A}\ \mathit{restart} \qquad \frac{}{\vdash B; \mathcal{N}, \neg A; A}\ \mathit{init}$$

$$\frac{\vdash B; \mathcal{N}; B_i}{\vdash B; \mathcal{N}; B_1 \vee B_2} \qquad \frac{}{\vdash B; \mathcal{N}; \mathit{true}} \qquad \frac{\vdash B; \mathcal{N}; B_1 \quad \vdash B; \mathcal{N}; B_2}{\vdash B; \mathcal{N}; B_1 \wedge B_2}$$

Here, A is an atom and $\mathcal{N}$ is a multiset of negated atoms. Sequents have three *zones*.

The $\vee$ rule can consume some external information or some non-determinism.

An *oracle string*, a series of bits used to indicate whether to go left or right, can be a proof certificate.

A proof in *LKpos*

Let C have several alternations of conjunction and disjunction.

Let $B = (p \vee C) \vee \neg p$.

$$
\begin{array}{cl}
\hline
\vdash B; \neg p; p & \textit{init} \\
\hline
\vdash B; \neg p; p \vee^{+} C & * \\
\hline
\vdash B; \neg p; (p \vee^{+} C) \vee^{+} \neg p & * \\
\hline
\vdash B; \cdot \quad ; \neg p & \textit{restart} \\
\hline
\vdash B; \cdot \quad ; (p \vee^{+} C) \vee^{+} \neg p & * \\
\hline
\vdash B & \textit{start}
\end{array}
$$

The subformula C is avoided. Clever choices $*$ are injected at these points: right, left, left. We have a small certificate and small checking time. In general, these certificates may grow large.

Combining the *LKneg* and *LKpos* proof systems

Introduce two versions of conjunction, disjunction, and their units.

$$t^-, t^+, f^-, f^+, \vee^-, \vee^+, \wedge^-, \wedge^+$$

Introduce the two kinds of sequent, namely,
$\vdash \Theta \Uparrow \Gamma$: for invertible (negative) rules (Γ a list of formulas)
$\vdash \Theta \Downarrow B$: for non-invertible (positive) rules (B a formula)

LKF : a focused proof systems for classical logic

$$\frac{}{\vdash \Theta \Uparrow \Gamma, t^-} \qquad \frac{\vdash \Theta \Uparrow \Gamma, B \qquad \vdash \Theta \Uparrow \Gamma, B'}{\vdash \Theta \Uparrow \Gamma, B \wedge^- B'} \qquad \frac{\vdash \Theta \Uparrow \Gamma}{\vdash \Theta \Uparrow \Gamma, f^-} \qquad \frac{\vdash \Theta \Uparrow \Gamma, B, B'}{\vdash \Theta \Uparrow \Gamma, B \vee^- B'}$$

$$\frac{}{\vdash \Theta \Downarrow t^+} \qquad \frac{\vdash \Theta \Downarrow B \qquad \vdash \Theta \Downarrow B'}{\vdash \Theta \Downarrow B \wedge^+ B'} \qquad \frac{\vdash \Theta \Downarrow B_i}{\vdash \Theta \Downarrow B \vee^+ B'}$$

$$\text{Init}\ \frac{}{\vdash \neg A, \Theta \Downarrow A} \qquad \text{Store}\ \frac{\vdash \Theta, C \Uparrow \Gamma}{\vdash \Theta \Uparrow \Gamma, C} \qquad \text{Release}\ \frac{\vdash \Theta \Uparrow N}{\vdash \Theta \Downarrow N} \qquad \text{Decide}\ \frac{\vdash P, \Theta \Downarrow P}{\vdash P, \Theta \Uparrow \cdot}$$

P is a positive formula; N is a negative formula;
A is an atom; C positive formula or negative literal

Results about LKF

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

Theorem. B is a tautology if and only if $\hat{B}$ has an LKF proof. [Liang & M, TCS 2009]

Thus the different polarizations do not change *provability* but can radically change the *proofs*.

Also:

- Negative (non-atomic) formulas are treated linearly (never weakened nor contracted).
- Only positive formulas are contracted (in the Decide rule).

An example

Assume that Θ contains the formula $a \wedge^+ b \wedge^+ \neg c$ and that we have a derivation that Decides on this formula.

$$
\dfrac{
\dfrac{}{\vdash \Theta \Downarrow a}\ \mathit{Init}
\qquad
\dfrac{}{\vdash \Theta \Downarrow b}\ \mathit{Init}
\qquad
\dfrac{\dfrac{\vdash \Theta, \neg c \Uparrow \cdot}{\vdash \Theta \Uparrow \neg c}}{\vdash \Theta \Downarrow \neg c}\ \mathit{Release}
}{
\dfrac{\vdash \Theta \Downarrow a \wedge^+ b \wedge^+ \neg c}{\vdash \Theta \Uparrow \cdot}\ \mathit{Decide}
}\ \mathit{and}
$$

This derivation is possible iff Θ is of the form $\neg a, \neg b, \Theta'$. Thus, the "macro-rule" is

$$
\frac{\vdash \neg a, \neg b, \neg c, \Theta' \Uparrow \cdot}{\vdash \neg a, \neg b, \Theta' \Uparrow \cdot}
$$

Example: Resolution as a proof certificate

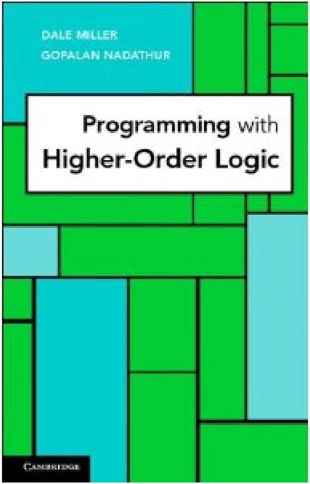
- A *clause:* $\forall x_1 \ldots \forall x_n [L_1 \vee \cdots \vee L_m]$
- C_3 is a *resolution* of C_1 and C_2 if we chose the mgu of two complementary literals, one from each of C_1 and C_2, etc.
- If C_3 is a resolvent of C_1 and C_2 then $\vdash \neg C_1, \neg C_2 \Uparrow C_3$ has a short proof (decide depth 2 or less).

Translate a refutation of $C_1, \ldots, C_n$ into a (focused) sequent proof with small holes:

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

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

Reference proof checking in λProlog



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.

From inference rules to λProlog clauses

We first "instrument" the inference rules with terms denoting proof certificates and add premises that invoke "clerks" and "experts".

$$\frac{\Xi_1 \vdash \Theta \Uparrow \Gamma, A \qquad \Xi_2 \vdash \Theta \Uparrow \Gamma, B \qquad \wedge\text{clerk}(\Xi_0, \Xi_1, \Xi_2)}{\Xi_0 \vdash \Theta \Uparrow \Gamma, A \wedge^- B}$$

$$\frac{\Xi_1 \vdash \Theta \Downarrow \Gamma, B_i \qquad \vee\text{expert}(\Xi_0, \Xi_1, i)}{\Xi_0 \vdash \Theta \Downarrow \Gamma, B_1 \vee^+ B_2}$$

Turning these inference rules sideways yields logic programs for which soundness is easy to show.

The formal definition of "proof evidence" involves

- describing the structure of the certificate terms Ξ and
- providing the definition of the clerk and expert predicates.

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.

Alone LF (a.k.a. λP) does not seem to have the right "atoms of inference."

- Canonical normal forms provide only one structuring of proofs (negative connectives and atoms).
- These lack an analytic notion of sharing and a natural treatments of parallel proof steps.

The multi-year ProofCert project

Recent results

- Formally define the FPC framework for first-order logic: for classical and for intuitionistic logics. Based on LJF and LKF (focused variant's of Gentzen's LJ and LK).
- Developed several proof certificate formats
  - Classical logic: expansion trees, matings, CNF, etc.
  - Intuitionistic logic: Frege systems, natural deduction, dependently typed λ-calculus, equality reasoning, etc.
- Implemented a reference kernel (using λProlog / Teyjus)

Some future plans

- Treat typed λ-calculi fully: LF, LFSC, λP-modulo (Deduki)
- Design many more FPCs: linear reasoning, DPLL, SAT, etc.
- Expand to handle proofs based on induction / co-induction / model checking. Need more proof theory for fixed points.
- Deployment. Competitions? TPTP?