The Proof-Theoretic Foundations of Logic Programming

Dale Miller

Inria Saclay & LIX, École Polytechnique
Palaiseau, France

Tease-LP, 28 May 2020

This talk is based on a paper that should be completed by the end of June. If you are interested in providing feedback on a draft, please contact me.

Roles of logic in computing

- *Computation-as-model:* Computation happens: registers change, lights flash, etc. Logic is used to make statements about what is happening.
  - classical or intuitionistic logic (or arithmetic) is typically used
  - also, various modal/temporal logics, Hoare triples
- *Computation-as-deduction:* Computation is based on bits of logic: formulas, terms, proofs.
  - *Proof normalization*, aka Curry-Howard correspondence. Foundations for functional programming.
  - *Proof search*. Foundations of logic programming, declarative databases, model checking, theorem proving, type inference, etc.

In this talk, I will focus on locating logic programming within this large domain of proof search.

Frameworks for logic programming: Desiderata

A good framework should have some desirable properties.

1. It should provide multiple and broad avenues for *reasoning* about programs. We don't need new versions of Turing machines: that is, we do not need another specification language that obviously computes but is hard to reason about.

2. It should allow for the proper *positioning* of the logic programming paradigm among other programming and specification paradigms.

3. It should provide for a range of *extensions*, leading to logic programming languages that go beyond the one acknowledged example based on Horn clauses.

Frameworks for logic programming: A history

Resolution refutations

- + Incorporates unification with inference
- - Refute, not prove. Skolemization. Restricted to classical logic.

Model theory

- + Mature mathematical basis
- - Valid if true in an infinite number of infinite models.

Operational semantics (transition systems, π-calculus)

- + Detailed formalization of operational behavior.
- - Complex metatheory, connection to logic distant.

Abstract machines

- + Possible to formalize mathematically.
- - About an implementation, connection to logic distant.

Structural proof theory: we examine this framework instead.

Problems in logic programming addressed by proof theory

In the 1980s, major foci of the community was on the two topics of *control* and *negation-as-failure*.

A few others were concerned with problems of accommodating programming language *abstractions*: i.e., modules, abstract datatypes, higher-order programming, etc.

With the work on λProlog during 1985-1994, the issues around abstraction were given a good treatment, both in proof theory and in implementations.

The topics of *control* and *negation-as-failure* were addressed later by more recent advances in proof theory.

Proof search uses of intuitionistic logic

In the second half of the 1980s, several researchers found important uses of intuitionistic logic within computation logic there were not directly related to the Curry-Howard correspondence. These discoveries were made nearly simultaneously and largely independently.

- Gabbay and Reyle, hypothetical, N-Prolog [1984 & 1985]
- Paulson - Isabelle generic (natural deduction, term-level binding) [1986 & 1989]
- M, Nadathur, Pfenning, & Scedrov - the λProlog effort, hypothetical and generic reasoning, term-level binding [1986-1989]
- Hallnäs and Schroeder-Heister - hypothetical reasoning, definitions [1990 & 1991]
- Thorne McCarty - hypothetical reasoning, intuitionistic negation [1988]

Proof theory

The term “proof theory” refers to different topics:

- ordinal analysis of proofs of *consistency* of a logic (via cut-elimination).
- *reverse mathematics* (choosing weak mathematical assumptions)
- analyzing the structure of proofs themselves.

This latter emphasis is often called “Structural Proof Theory”. See such authors as

- Gentzen
- Gallier: “Logic for Computer Science” [1986]
- Girard, Taylor, & Lafont: “Proofs and Types” [1989]
- Negri and von Plato: “Structural Proof Theory” [2001]

The sequent calculus

Introduced by Gentzen as an explicit attempt to provide a unifying framework for proofs in classical and intuitionistic logics.

Girard's linear logic can be further unified with these logics via the sequent calculus.

- Proofs without cuts are *analytic* (involving only subformulas): there are no cut-free proofs of false.
- Introducing the cut rule meant that all of first-order logic was captured. Cut-elimination was a kind of completeness result: analytic proofs were sufficient.

A personal reflection.

Unity of logic

LJ proofs are LK proofs in which the right-hand side has at most one formula.

$\Gamma \vdash \Delta$ (multiple conclusion) *vs.* $\Gamma \vdash B$ (single conclusion)

Gentzen's sequent calculus work was an early attempt at a "unity of logic". Making structural rules explicit—especially on the right—is critical.

Girard's linear logic makes structural rules also explicit on the left.

Sequent calculus is an appealing tool to study proof theory and computational logic since it captures and relates these three logics.

Applications of sequent calculus

Sequent calculus has been used in:

Proof theory

- Consistency of logic and arithmetic
- Herbrand's theorem
- Midsequent theorem
- Interpolation
- Negative translations

Computer Science

- foundations for logic programming:
- explicit substitutions in the λ-calculus
- syntactic correctness of Skolemization
- etc.

However: Sequent calculus proofs are chaotic, painful, etc

Sequent calculus proofs are formless.

Any structure they might contain about a proof needs to be pulled out by extensive inference-rule permutation arguments.

It is common to say that

- a sequent calculus proof is a *computation of a proof* while
- a natural deduction proof is the *actual proof*.

We will be more sophisticated than that in this talk: we improve on sequent calculus by moving to *focused* proofs (instead of natural deduction).

Permutation of inference rules illustrated

Let Γ be a multiset of 998 formulas and consider searching for a proof of the sequent $\Gamma, B_1 \vee B_2, C_1 \wedge C_2 \vdash A$.

There are 1000 choices for the left introduction rule to attempt this proof and the resulting premises can be attempted using 1000 left-introduction rules.

For example,

$$\dfrac{\dfrac{\Gamma, B_1, C_1, C_2 \vdash A}{\Gamma, B_1, C_1 \wedge C_2 \vdash A}\,\wedge L \quad \dfrac{\Gamma, B_2, C_1, C_2 \vdash A}{\Gamma, B_2, C_1 \wedge C_2 \vdash A}\,\wedge L}{\Gamma, B_1 \vee B_2, C_1 \wedge C_2 \vdash A}\,\vee L$$

is one in about a million choices.

- Another choice switches the order of $\vee L$ and $\wedge L$: that switch is **not important**.
- Also these two inference rules are *invertible* and can be applied automatically; without needing to reconsider them.

Problems with the sequent calculus

Inference rules in the sequent calculus are

- too tiny,
- too independent from each other, and
- not the right inference rules is many settings.

To illustrate the last point, it is more common to need inference rules such as one of the following pairs of rules.

$$\frac{\Gamma \vdash \mathit{adj}\ x\ y}{\Gamma \vdash \mathit{path}\ x\ y} \qquad \frac{\Gamma \vdash \mathit{path}\ x\ z \quad \Gamma \vdash \mathit{path}\ z\ y}{\Gamma \vdash \mathit{path}\ x\ y}$$

$$\frac{\Gamma, \mathit{adj}\ x\ y, \mathit{path}\ x\ y \vdash A}{\Gamma, \mathit{adj}\ x\ y \vdash A} \qquad \frac{\Gamma, \mathit{path}\ x\ z, \mathit{path}\ z\ y, \mathit{path}\ x\ y \vdash A}{\Gamma, \mathit{path}\ x\ z, \mathit{path}\ z\ y \vdash A}$$

NB: these rules contain no occurrences of logical connectives.

LJF for only $\supset$ and $\forall$

LJF is a focused version of Gentzen's **LJ** proof system for intuitionistic logic introduced by Liang & M [TCS 2009].

There are three kinds of sequents

1. unfocused sequents: $\Gamma \vdash B$
2. left focused sequent: $\Gamma \Downarrow B \vdash A$, with focus B
3. right focused sequent: $\Gamma \vdash A \Downarrow$, with focus A

Replacing $\Uparrow$ on the left with a comma yields a regular sequent.

Γ as a syntactic variable ranging over multisets of formulas.
A as a syntactic variable ranging over atomic formulas.

A subset of **LJF**: the introduction rules

The right introduction rules for $\supset$ and $\forall$ are invertible.

$$\frac{\Gamma, B_1 \vdash B_2}{\Gamma \vdash B_1 \supset B_2} \qquad\qquad \frac{\Gamma \vdash [y/x]B}{\Gamma \vdash \forall x.B}\ y \text{ not free in conclusion}$$

The left introduction rules for $\supset$ and $\forall$ are not invertible.

$$\frac{\Gamma \Downarrow [t/x]B \vdash A}{\Gamma \Downarrow \forall x.B \vdash A} \qquad \frac{\Gamma \vdash B_1 \Downarrow \qquad \Gamma \Downarrow B_2 \vdash A}{\Gamma \Downarrow B_1 \supset B_2 \vdash A}$$

LJF: Polarity and the remaining inference rules

Polarity (simplified)

1. Formulas of the form $B_1 \supset B_2$ and $\forall x.B$ are *negative*.
2. Atomic formulas can be given arbitrary polarity: all *negative*; all *positive*; or a mixture of both.

Decide:
$$\frac{\Gamma, N \Downarrow N \vdash A}{\Gamma, N \vdash A}\ D_l \qquad \frac{\Gamma \vdash P \Downarrow}{\Gamma \vdash P}\ D_r$$

Release:
$$\frac{\Gamma, P \vdash A}{\Gamma \Downarrow P \vdash A}\ R_l \qquad \frac{\Gamma \vdash N}{\Gamma \vdash N \Downarrow}\ R_r$$

Initial:
$$\frac{N \text{ atomic}}{\Gamma \Downarrow N \vdash N}\ I_l \qquad \frac{P \text{ atomic}}{\Gamma, P \vdash P \Downarrow}\ I_r$$

Here, P is a positive (atomic) formula and N is a negative formula.

Formal results about **LJF**

Theorem: Let B be a first-order formula over $\forall$ and $\supset$.

- If $\vdash B$ is provable in **LJ** then for every polarization of atomic formulas, the sequent $\vdash B$ is provable in **LJF**.
- If atoms are given some polarization and $\vdash B$ is provable in **LJF**, then $\vdash B$ is provable in **LJ**.

Proof: Follows from a result in Liang & M [TCS 2009].

Polarization of atoms does not affect provability but can make a big impact on the structure of proofs.

Uniform proofs correspond to the case where all atomic formulas are polarized negatively. The notion of “uniform proof” should now be replaced with the richer notion of “focused proof”.

Characterizing forward and backward-chaining

$$\dfrac{\dfrac{\begin{array}{c}\Xi_1\\ \Gamma \vdash Rab \Downarrow\end{array} \quad \dfrac{\begin{array}{c}\Xi_2\\ \Gamma \vdash Rbc \Downarrow\end{array} \quad \begin{array}{c}\Xi_3\\ \Gamma \Downarrow Rac \vdash A\end{array}}{\Gamma \Downarrow Rbc \supset Rac \vdash A}\supset L}{\Gamma \Downarrow Rab \supset Rbc \supset Rac \vdash A}\supset L}{\Gamma \Downarrow \forall x \forall y \forall z (Rxy \supset Ryz \supset Rxz) \vdash A}\forall L \times 3$$

If atoms have neg polarity, then Ξ_3 is initial and A is Rac. Also, Ξ_1 and Ξ_2 are release. The synthetic rule is *backward-chaining*.

$$\dfrac{\Gamma \vdash Rab \quad \Gamma \vdash Rbc}{\Gamma \vdash Rac}$$

If atoms have pos polarity, then Ξ_3 is release and Ξ_1, Ξ_2 are initial and Γ is Rab, Rbc, Γ'. The synthetic rule is *forward-chaining*.

$$\dfrac{\Gamma', Rab, Rbc, Rac \vdash A}{\Gamma', Rab, Rbc \vdash A}$$

See Chaudhuri, Pfenning, and Price [JAR 2008].

Another example: Fibonacci numbers

Let Γ contain $\mathit{fib}(0,0)$, $\mathit{fib}(1,1)$, and

$$\forall n \forall f \forall f' [\mathit{fib}(n,f) \supset \mathit{fib}(n+1,f') \supset \mathit{fib}(n+2,f+f')].$$

The nth Fibonacci number is F iff $\Gamma \vdash \mathit{fib}(n,F)$.

backward-chaining: $$\frac{\Gamma \vdash \mathit{fib}(n,f) \qquad \Gamma \vdash \mathit{fib}(n+1,f')}{\Gamma \vdash \mathit{fib}(n+2,f+f')}$$

forward-chaining: $$\frac{\Gamma, \mathit{fib}(n,f), \mathit{fib}(n+1,f'), \mathit{fib}(n+2,f+f') \vdash A}{\Gamma, \mathit{fib}(n,f), \mathit{fib}(n+1,f') \vdash A}$$

If the polarity of atoms is

- negative, the unique proof is *exponential* in n.
- positive, the shortest proof is *linear* in n.

Synthetic inference rules

In Kowalski's paper [CACM 1979] *Algorithm = Logic + Control*, two control regimes were described, corresponding to forward-chaining and backward-chaining. We can now explain them proof theoretically using focused proof systems.

Negri's result [AML 2003] concerning converting axioms to inference rules can also be explained and generalized.

More generally, synthetic rules built using focusing automatically satisfy cut-elimination. See Marin, M, Pimentel, & Volpe [2020].

Synthetic inference rules: another example

In set theory, the following implication relates the subset and membership predicates:

$$\forall yz.(\forall x(x \in y \supset x \in z) \supset y \subseteq z).$$

If these predicates are polarized positively, the synthetic inference rule is

$$\frac{x \in y, \Gamma \vdash x \in z \qquad y \subseteq z, \Gamma \vdash E}{\Gamma \vdash E}\ .$$

If these predicates are polarized negatively, the synthetic inference rule is

$$\frac{x \in y, \Gamma \vdash x \in z}{\Gamma \vdash y \subseteq z}\ .$$

In both cases, x is an eigenvariable for that rule.

Another success of proof theory: computing with bindings

Encode untyped lambda-terms sending bindings to bindings:

$$\lceil \lambda x.B \rceil = (abs\ (\lambda x.\lceil B \rceil)).$$

The following formula specifies the typing of a λ-abstraction.

$$\forall B \forall \tau \forall \tau' [\forall x (of\ x\ \tau \supset of\ (Bx)\ \tau') \supset of\ (abs\ B)\ (\tau \to \tau')].$$

Now consider the following combination of inference rules.

$$\cfrac{\cfrac{\Sigma, x : \Delta, of\ \lceil x \rceil\ \tau \vdash of\ \lceil B \rceil\ \tau'}{\Sigma : \Delta \vdash \forall x\ (of\ \lceil x \rceil\ \tau \supset of\ \lceil B \rceil\ \tau')}\ \forall R,\ \supset R}{\Sigma : \Delta \vdash of\ \lceil \lambda x.B \rceil\ (\tau \to \tau')}\ \text{backchaining}$$

The binding for x moves from the term-level, to the formula-level (as a quantifier), to the proof-level (as an eigenvariable).

A binding is not converted to a "free variable": it simply moves.

Binder mobility

Binder mobility does not exist in *higher-order Horn clauses*.
λProlog was extended with hypothetical and generic ($\forall$) goals.

The Abella theorem prover [Baelde et al. 2014] enhances binder mobility by providing new binding sites: the ∇-quantifier at the formula-level and nominal contexts at the proof-level.

Incorporating binder mobility into unification avoids the need for Skolemization to simplify quantifier alternations.

$$\begin{aligned} \forall x \exists y \forall z\ &[(f\ z\ y) = (f\ z\ x)] \\ \forall x \exists y\ &[\lambda z (f\ z\ y) = \lambda z (f\ z\ x)] \quad (\xi) \\ \exists h \forall x\ &[\lambda z (f\ z\ (h\ x)) = \lambda z (f\ z\ x)] \quad (\mathit{raising}) \\ \exists h\ &[\lambda x \lambda z (f\ z\ (h\ x)) = \lambda x \lambda z (f\ z\ x)] \quad (\xi) \end{aligned}$$

See *unification under a mixed prefix* [M, 1992].

Negation-as-failure

The proof-theoretic solution arose from simultaneous and independent results

- Hallnäs & Schroeder-Heister [JLC 1991], definitions
- Girard [1992], fixpoints

These results show that *finite failure* can be formalized into an inference rule for negation.

Their work has been generalize to a new approach to checking simulation/bisimulation and induction/coinduction in proof theory [McDowell 1997; Tiu 2004; Gacek 2009; Baelde 2012].

It provides a proof theory foundation for model checking [Heath & M, 2019].

Linear Logic

We owe the notions of polarization and focusing to linear logic [Andreoli 1992]. The depth of these ideas would not have been clear from studying intuitionistic and classical logics alone.

Many *linear logic programming languages* have been developed to support

- state changes,
- concurrency, and
- multiset rewriting.

Linear logic programming, especially when exploiting *subexponentials*, has successes as a logical framework for specifying other proof systems [Nigam, Olarte, Pimentel, Reis, etc]

Some near-term prospects for logic programming

Automating Operational Semantics

Bring back the early project by Gilles Kahn and his team from the late 1980s to automate interpreters, debuggers, compilers, static checkers, etc based on big step operational semantics. See the Makam implementation of λProlog [Stampoulis & Chlipala, 2018].

Proof checking

Logic programs can do some degree of *proof reconstruction* and *proof elaboration* when checking proof certificates.

Proof refinement in interactive theorem provers

Use logic programming engines to power the tactic (proof refinement). Some development work has started with the ELPI implementation of λProlog and its integration into the Coq prover [Tassi 2020].

Formalized reasoning about logic programs

A team of us are just getting started with using Abella to reason about logic programs and logical specifications.

Bibliography: selected references

- J.-M. Andreoli. Logic programming with focusing proofs in linear logic. *J. of Logic and Computation*, 2(3):297–347, 1992.
- D. Baelde, K. Chaudhuri, A. Gacek, D. Miller, G. Nadathur, A. Tiu, and Y. Wang. Abella: A system for reasoning about relational specifications. *Journal of Formalized Reasoning*, 7(2):1–89, 2014.
- K. Chaudhuri, F. Pfenning, and G. Price. A logical characterization of forward and backward chaining in the inverse method. *J. of Automated Reasoning*, 40(2-3):133–177, Mar. 2008.
- D. Clément, J. Despeyroux, T. Despeyroux, L. Hascoët, and G. Kahn. Natural semantics on the computer. Research Report 416, INRIA, Rocquencourt, France, June 1985.
- A. Gacek, D. Miller, and G. Nadathur. A two-level logic approach to reasoning about computations. *J. of Automated Reasoning*, 49(2):241–273, 2012.
- Q. Heath and D. Miller. A proof theory for model checking. *J. of Automated Reasoning*, 63(4):857–885, 2019.
- C. Liang and D. Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theoretical Computer Science*, 410(46):4747–4768, 2009.
- D. Miller. Unification under a mixed prefix. *Journal of Symbolic Computation*, 14(4):321–358, 1992.
- D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51(1–2):125–157, 1991.
- S. Negri. Contraction-free sequent calculi for geometric theories with an application to Barr's theorem. *Archive for Mathematical Logic*, 42:389–401, 2003.
- V. Nigam, E. Pimentel, and G. Reis. An extended framework for specifying and reasoning about proof systems. *J. of Logic and Computation*, 2014.
- P. Schroeder-Heister. Rules of definitional reflection. In M. Vardi, editor, *8th Symp. on Logic in Computer Science*, pages 222–232. IEEE Computer Society Press, IEEE, June 1993.
- E. Tassi. ELPI: an extension language for Coq. CoqPL 2018: The Fourth International Workshop on Coq for Programming Languages, Jan. 2018.