Proof Theory and Type Theory: Distinct Foundations for Designing Proof Assistants

Dale Miller

Inria Saclay &
LIX, Institut Polytechnique de Paris
Partout Team

Banff meeting, June 2025

Art by Nadia Miller

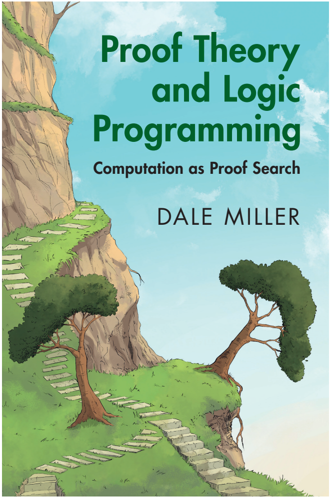
Proof Theory and Logic Programming: Computation as proof search, by Dale Miller

To be published by Cambridge University Press by December 2025.

Preprint available from my web page. `https://www.lix.polytechnique.fr/Labo/Dale.Miller/ptlp/`

Organizes everything I learned about the intersection of proof theory and logic programming during four decades (1985-2025).

Uses classical, intuitionistic, and linear logic (first-order and higher-order) to design and reason about logic programs.



Art by Nadia Miller

Outline

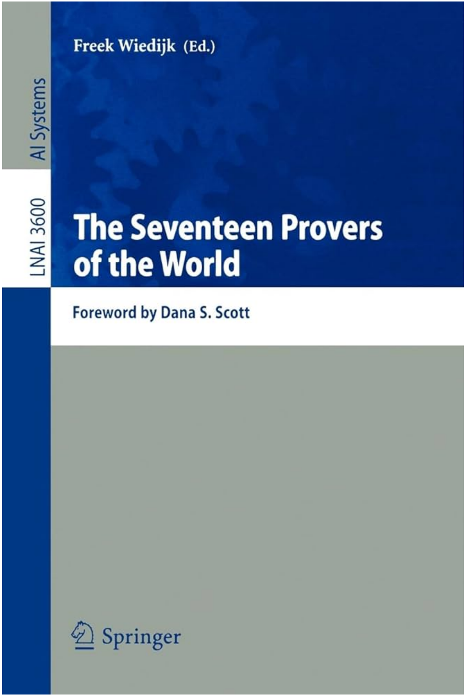
The world of proof assistants

The Abella proof assistant

A proof theorist's view of type theory

Sequents and binders

Back to Rocq vs Abella



Published in 2006.

List of proof assistants by formalisms

- Church's STT of types: HOL, Isabelle/Isar, IMPS, Ωmega, Minlog, Theorema (Frege proofs with discharge)
- First-order logic: Mizar, Otter/Ivy, ACL2, B Method, Metamath (Frege proofs, resolution, equality reasoning)
- Type Theory (dependently typed λ-terms): Coq, Alfa/Agda, Lego, Nuprl, PVS, PhoX

List of proof assistants by formalisms

- Church's STT of types: HOL, Isabelle/Isar, IMPS, Ωmega, Minlog, Theorema (Frege proofs with discharge)
- First-order logic: Mizar, Otter/Ivy, ACL2, B Method, Metamath (Frege proofs, resolution, equality reasoning)
- Type Theory (dependently typed λ-terms): Coq, Alfa/Agda, Lego, Nuprl, PVS, PhoX

While this classification might be debated, it is clear that

no system is based on Structural Proof Theory.

List of proof assistants by formalisms

- Church's STT of types: HOL, Isabelle/Isar, IMPS, Ωmega, Minlog, Theorema (Frege proofs with discharge)
- First-order logic: Mizar, Otter/Ivy, ACL2, B Method, Metamath (Frege proofs, resolution, equality reasoning)
- Type Theory (dependently typed λ-terms): Coq, Alfa/Agda, Lego, Nuprl, PVS, PhoX

While this classification might be debated, it is clear that

no system is based on Structural Proof Theory.

I take this as a major professional challenge, since I've been using such proof theory successfully with

- Logic programming: e.g., Prolog, λProlog, linear logic programming, and
- Model checking: Bedwyr [J. Automated Deduction, 2019]

Outline

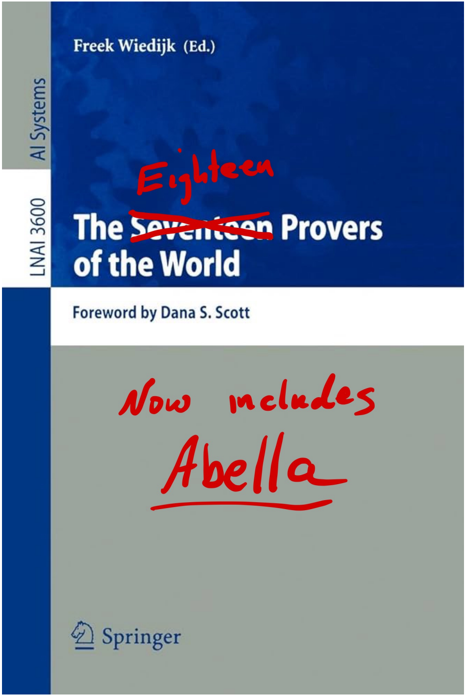
The world of proof assistants

The Abella proof assistant

A proof theorist's view of type theory

Sequents and binders

Back to Rocq vs Abella



Abella appeared in 2009.

Abella

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
- Several theorems about the π-calculus
- 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.

Inside Abella

The previous slide is the public face of Abella. The real story is that Abella is an implementation of the sequent calculus. A goal in Abella is displayed as

```
Variables: x1 ... xm
H1 : A1
...
Hn : An
==============================
 C
```

where $x_1, \ldots, x_m$ are universally quantified variables, `H1`, ..., `Hn` are hypothesis labels that are each associated with a unique hypothesis formula drawn from $A_1, \ldots, A_n$ and C is a formula called the conclusion of the goal. The collection of variables and hypotheses is called the context of the goal. This goal denotes, of course, the sequent

$$x_1 : \tau_1, \ldots, x_m : \tau_m :: A_1, \ldots, A_n \vdash C.$$

Example sequents in Abella

```
Variables: M N
IH: forall M N, nat M * -> plus M M N -> times
    (s^2 z) M N
H1: nat M @
H2: plus M M N
  ============================
 times (s^2 z) M N
```

Example sequents in Abella

```
Variables: M N
IH: forall M N, nat M * -> plus M M N -> times
   (s^2 z) M N
H1: nat M @
H2: plus M M N
  ============================
 times (s^2 z) M N
```

```
Variables: N1 K1
IH: forall M N, nat M * -> plus M M N -> times
   (s^2 z) M N
H3: nat N1 *
H4: plus N1 (s N1) (s K1)
H6: plus N1 N1 K1
H7: times (s^2 z) N1 K1
  ============================
 plus (s N1) (s N1) (s^2 K1)
```

Papers developing the proof theory of Abella

Start with Gentzen's LJ. Add the following extensions.

Equality: Girard [1992], Schroeder-Heister [LICS 1993], McDowell & M [TCS 2000], M & Viel [AMAI 2022]

Fixed points: McDowell & M [TCS 2000], Momigliano & Tiu [JAL 2012], Baelde [ToCL 2012]

These raise first-order logic to Heyting Arithmetic.

Papers developing the proof theory of Abella

Start with Gentzen's LJ. Add the following extensions.

Equality: Girard [1992], Schroeder-Heister [LICS 1993], McDowell & M [TCS 2000], M & Viel [AMAI 2022]

Fixed points: McDowell & M [TCS 2000], Momigliano & Tiu [JAL 2012], Baelde [ToCL 2012]

These raise first-order logic to Heyting Arithmetic.

∇-quantification: Tiu & M [LICS 2003]

Nominal-abstraction: Gacek, M, & Nadathur [I&C 2011]

These additions yield the $\mathcal{G}$ logic. It is here that the <u>λ-tree syntax</u> approach to binders appears.

Outline

A proof theorist's view of type theory: The structure of proof

Type Theory generally settles two questions simultaneously:

- Which logic are you using? Typically, intuitionistic logic.
- What is a proof? Typically, natural deduction proofs encoded as dependently, typed λ-terms.

A proof theorist's view of type theory: The structure of proof

Type Theory generally settles two questions simultaneously:

- Which logic are you using? Typically, intuitionistic logic.
- What is a proof? Typically, natural deduction proofs encoded as dependently, typed λ-terms.

A proof theorist usually separates these questions and allows a wide range of proof systems.

Intuitionistic logic: sequent calculus, natural deduction, tableaux

Classical logic: sequent calculus, tableaux, expansion trees, resolution refutations, natural deduction with restart

Linear logic: proof nets, as well.

Deep inference structures are also generally applicable.

A proof theorist's view of type theory: Structural problems with the proof-as-λ-term approach

λ-reduction is

- wildly non-deterministic, resulting in CBV, CBN, CBPV, etc, and
- not the most efficient way to normalize expressions.

A proof theorist's view of type theory: Structural problems with the proof-as-λ-term approach

λ-reduction is

- wildly non-deterministic, resulting in CBV, CBN, CBPV, etc, and
- not the most efficient way to normalize expressions.

Typed λ-terms

- can be highly redundant structures and
- have no explicit structure-sharing mechanisms.

A proof theorist's view of type theory: Structural problems with the proof-as-λ-term approach

λ-reduction is

- wildly non-deterministic, resulting in CBV, CBN, CBPV, etc, and
- not the most efficient way to normalize expressions.

Typed λ-terms

- can be highly redundant structures and
- have no explicit structure-sharing mechanisms.

Other complicating features:

- proof irrelevance: too many subproofs kept
- implicit arguments: too inconvenient to supply all arguments
- universe levels: needed to organize rich typing structures

A proof theorist's view of type theory: Many aspects of type theory come from proof theory

- Cut-elimination, non-atomic initial elimination: these give rise to β and η-conversions.
- Cut-elimination is called weak normalization in type theory. Strong normalization is often of secondary importance in proof theory.
- Canonical dependently typed λ-terms derived from the notion of uniform proofs (focused proofs).
- Linear logic appears in proof theory first and later moves to type theory.

A proof theorist's view of type theory: The type theory approach to classical logic

Gentzen [1935] added the excluded middle to NJ to get NK (natural deduction for classical logic). He abandoned NK since it did not have good proof-theoretic properties.

Unfortunately, Gentzen's solution (multiple-conclusion sequent calculus) is ruled out by type theory (generally speaking).

The Rocq system allows classical reasoning but only via the addition of appropriate axioms.

A proof theorist's view of type theory: Treatment of bindings

Higher-order abstract syntax: If your object-level syntax (formulas, programs, types, etc) contain binders, then map them to meta-language binders.

Type theory: the binders available are those for function spaces.

Proof Search: the binders available are λ-expressions with equality modulo λ-conversions (as in λProlog).

These approaches are different. Consider $\forall w_i.\ \lambda x.x \neq \lambda x.w$.

A proof theorist's view of type theory: Treatment of bindings

Higher-order abstract syntax: If your object-level syntax (formulas, programs, types, etc) contain binders, then map them to meta-language binders.

Type theory: the binders available are those for function spaces.

Proof Search: the binders available are λ-expressions with equality modulo λ-conversions (as in λProlog).

These approaches are different. Consider $\forall w_i.\ \lambda x.x \neq \lambda x.w$.

Type theory: Not a theorem since the identity and the constant valued function coincide on singleton domains.

A proof theorist's view of type theory: Treatment of bindings

Higher-order abstract syntax: If your object-level syntax (formulas, programs, types, etc) contain binders, then map them to meta-language binders.

Type theory: the binders available are those for function spaces.

Proof Search: the binders available are λ-expressions with equality modulo λ-conversions (as in λProlog).

These approaches are different. Consider $\forall w_i.\ \lambda x.x \neq \lambda x.w$.

Type theory: Not a theorem since the identity and the constant valued function coincide on singleton domains.

Proof search: Is a theorem since no instance of $\lambda x.w$ equals $\lambda x.x$.

A proof theorist's view of type theory: Treatment of bindings

Higher-order abstract syntax: If your object-level syntax (formulas, programs, types, etc) contain binders, then map them to meta-language binders.

Type theory: the binders available are those for function spaces.

Proof Search: the binders available are λ-expressions with equality modulo λ-conversions (as in λProlog).

These approaches are different. Consider $\forall w_i.\ \lambda x.x \neq \lambda x.w$.

Type theory: Not a theorem since the identity and the constant valued function coincide on singleton domains.

Proof search: Is a theorem since no instance of $\lambda x.w$ equals $\lambda x.x$.

The latter approach to HOAS is called the λ-tree syntax approach.

How can proof theory account for binders?

Outline

Dynamics of binders during proof search

During computation, binders can be instantiated

$$\frac{\Sigma : \Gamma, \mathit{typeof}\ c\ (\mathit{int} \to \mathit{int}) \vdash C}{\Sigma : \Gamma, \forall\alpha(\mathit{typeof}\ c\ (\alpha \to \alpha)) \vdash C}\ \forall L$$

Dynamics of binders during proof search

During computation, binders can be instantiated

$$\frac{\Sigma : \Gamma, \mathit{typeof}\ c\ (\mathit{int} \to \mathit{int}) \vdash C}{\Sigma : \Gamma, \forall\alpha(\mathit{typeof}\ c\ (\alpha \to \alpha)) \vdash C}\ \forall L$$

or they can move (a feature called the mobility of binders):

$$\cfrac{\cfrac{\Sigma, \mathbf{x} : \Gamma, \mathit{typeof}\ x\ \alpha \vdash \mathit{typeof}\ \lceil B \rceil\ \beta}{\Sigma : \Gamma \vdash \forall\mathbf{x}(\mathit{typeof}\ x\ \alpha \supset \mathit{typeof}\ \lceil B \rceil\ \beta)}\ \forall R}{\Sigma : \Gamma \vdash \mathit{typeof}\ \lceil \lambda\mathbf{x}.B \rceil\ (\alpha \to \beta)}$$

The binder for x moves from term-level (λx) to formula-level ($\forall x$) to proof-level (as an eigenvariable).

Note: The variables in the signature Σ are eigenvariables and are bound over $\Gamma \vdash C$.

"There is no such thing as a free variable."-Epigram 47, A. Perlis

Quiz

Consider a simple object-logic with a pairing constructor $\langle x, y \rangle$.

Assume that the formula $\forall u \forall v[q\ \langle u, t_1 \rangle\ \langle v, t_2 \rangle\ \langle v, t_3 \rangle]$ follows from the assumptions

$$L = \{\forall x \forall y[q\ x\ x\ y],\ \forall x \forall y[q\ x\ y\ x],\ \forall x \forall y[q\ y\ x\ x]\}.$$

What can we say about the terms t_1, t_2, and t_3?

Quiz

Consider a simple object-logic with a pairing constructor $\langle x, y \rangle$.

Assume that the formula $\forall u \forall v [q\ \langle u, t_1 \rangle\ \langle v, t_2 \rangle\ \langle v, t_3 \rangle]$ follows from the assumptions

$$L = \{\forall x \forall y [q\ x\ x\ y],\ \forall x \forall y [q\ x\ y\ x],\ \forall x \forall y [q\ y\ x\ x]\}.$$

What can we say about the terms t_1, t_2, and t_3?

Answer: The terms t_2 and t_3 are equal. We would like to prove (and we can prove in Abella)

$$\forall t_1 \forall t_2 \forall t_3 [prv\ L\ (\forall u \forall v [q\ \langle u, t_1 \rangle\ \langle v, t_2 \rangle\ \langle v, t_3 \rangle]) \supset t_2 = t_3$$

This conclusion holds for intensional, not extensional, reasons.

Such an intensional treatment does not seem possible if binder mobility to the proof level is limited to eigenvariables.

Generic judgments and the ∇-quantifier

We add to sequents another binding context: attach a local signature to every formula.

$$\Sigma : B_1, \ldots, B_n \longrightarrow B_0$$
$$\Downarrow$$
$$\Sigma : \sigma_1 \triangleright B_1, \ldots, \sigma_n \triangleright B_n \longrightarrow \sigma_0 \triangleright B_0$$

Here, σ_i is a list of distinct variables scoped over B_i.

The expression $\sigma_i \triangleright B_i$ is called a generic judgment.

Generic judgments and the ∇-quantifier

We add to sequents another binding context: attach a local signature to every formula.

$$\Sigma : B_1, \ldots, B_n \longrightarrow B_0$$
$$\Downarrow$$
$$\Sigma : \sigma_1 \triangleright B_1, \ldots, \sigma_n \triangleright B_n \longrightarrow \sigma_0 \triangleright B_0$$

Here, σ_i is a list of distinct variables scoped over B_i.

The expression $\sigma_i \triangleright B_i$ is called a generic judgment.

Standard proof theory design: Enrich with a new context and add connectives that deal with that enrichment.

Generic judgments and the ∇-quantifier

We add to sequents another binding context: attach a local signature to every formula.

$$\Sigma : B_1, \ldots, B_n \longrightarrow B_0$$
$$\Downarrow$$
$$\Sigma : \sigma_1 \triangleright B_1, \ldots, \sigma_n \triangleright B_n \longrightarrow \sigma_0 \triangleright B_0$$

Here, σ_i is a list of distinct variables scoped over B_i.

The expression $\sigma_i \triangleright B_i$ is called a generic judgment.

Standard proof theory design: Enrich with a new context and add connectives that deal with that enrichment.

The left and right introductions for ∇ (nabla) are

$$\frac{\Sigma : (\sigma, x : \tau) \triangleright B, \Gamma \longrightarrow \mathcal{C}}{\Sigma : \sigma \triangleright \nabla_\tau x.B, \Gamma \longrightarrow \mathcal{C}} \qquad \frac{\Sigma : \Gamma \longrightarrow (\sigma, x : \tau) \triangleright B}{\Sigma : \Gamma \longrightarrow \sigma \triangleright \nabla_\tau x.B}$$

The remaining proof theory can be explored directly

Note that ∇ is self-dual: $\neg\nabla x.Bx \dashv\vdash \nabla x.\neg Bx$

How does ∇ interact with the other connectives and quantifiers?

How do we add induction and coinduction?

When are two generic judgments equal?

The remaining proof theory can be explored directly

Note that ∇ is self-dual: $\neg\nabla x.Bx \dashv\vdash \nabla x.\neg Bx$

How does ∇ interact with the other connectives and quantifiers?

How do we add induction and coinduction?

When are two generic judgments equal?

Cut-elimination - mostly follows the structure of first-order intuitionistic logic.

A classical logic treatment is straightforward: the difference between open and late bisimulation in the π-calculus is a choice of choosing intuitionistic or classical logic.

Abella was developed on these proof theory results.

Applications of ∇ to meta-theory

π-calculus

- A Proof Theory for Generic Judgments, by M and Tiu, ToCL 2005.
- Proof search specifications of bisimulation and modal logics for the π-calculus by Tiu and M. ToCL, 2010.
- A lightweight formalization of the metatheory of bisimulation-up-to by Chaudhuri, Cimini, and M. CPP 2015.

Applications of ∇ to meta-theory

π-calculus

- A Proof Theory for Generic Judgments, by M and Tiu, ToCL 2005.
- Proof search specifications of bisimulation and modal logics for the π-calculus by Tiu and M. ToCL, 2010.
- A lightweight formalization of the metatheory of bisimulation-up-to by Chaudhuri, Cimini, and M. CPP 2015.

λ-calculus

- Proof pearl: Abella formalization of lambda calculus cube property, by Accattoli CPP 2012.
- A Mechanical Formalization of Higher-Ranked Polymorphic Type Inference, by Zhao, Oliveira, and Schrijvers, ICFP 2019.
- Barendregt's theory of the lambda-calculus, refreshed and formalized, by Lancelot, Accattoli, and Vemclefs. ITP 2025.

Applications of ∇ to meta-theory

π-calculus

- A Proof Theory for Generic Judgments, by M and Tiu, ToCL 2005.
- Proof search specifications of bisimulation and modal logics for the π-calculus by Tiu and M. ToCL, 2010.
- A lightweight formalization of the metatheory of bisimulation-up-to by Chaudhuri, Cimini, and M. CPP 2015.

λ-calculus

- Proof pearl: Abella formalization of lambda calculus cube property, by Accattoli CPP 2012.
- A Mechanical Formalization of Higher-Ranked Polymorphic Type Inference, by Zhao, Oliveira, and Schrijvers, ICFP 2019.
- Barendregt's theory of the lambda-calculus, refreshed and formalized, by Lancelot, Accattoli, and Vemclefs. ITP 2025.

Abella developments are small: there is not yet a lot of support for big developments.

Specifying object-level provability as an inductive predicate

```
seq Hs A               := memb A L.
seq Hs A               := prog A B /\ seq Hs B.
seq Hs (B and C)       := seq Hs B /\ seq Hs C.
seq Hs (B imp C)       := seq (B::Hs) C.
```

```
prog (plus z N N) tt.
prog (plus (s M) N (s P)) (plus M N P).
  ...
```

Specifying object-level provability as an inductive predicate

```
seq Hs A              := memb A L.
seq Hs A              := prog A B /\ seq Hs B.
seq Hs (B and C)      := seq Hs B /\ seq Hs C.
seq Hs (B imp C)      := seq (B::Hs) C.
```

```
prog (plus z N N) tt.
prog (plus (s M) N (s P)) (plus M N P).
  ...
```

Theorems:

```
forall N M, seq nil (plus N M N)     -> M = z.
forall N,   seq nil (plus (s z) N z) -> false
```

Specifying object-level provability as an inductive predicate

```
seq Hs A                := memb A L.
seq Hs A                := prog A B /\ seq Hs B.
seq Hs (B and C)        := seq Hs B /\ seq Hs C.
seq Hs (B imp C)        := seq (B::Hs) C.
```

```
prog (plus z N N) tt.
prog (plus (s M) N (s P)) (plus M N P).
  ...
```

Theorems:

```
forall N M, seq nil (plus N M N)      -> M = z.
forall N,   seq nil (plus (s z) N z) -> false
```

Treating object-level eigenvariables

```
seq Hs (all x\ B x) := nabla x\ seq Hs (B x).
```

Outline

The many approaches to binders used with Rocq

Many packages have been implemented, and none seem canonical.

- Named Variables (first-order)
- De Bruijn Indices
- Locally Nameless
- Higher-Order Abstract Syntax (HOAS)
- Hybrid: A Definitional Two-Level Approach to Reasoning with HOAS (Felty & Momigliano)
- Parametric HOAS (PHOAS)
- Nominal Approach (Pitts, Gabbay, etc)

There are challenge problems (POPLMark, POPLMark reloaded), case studies, benchmarks, and surveys.

Different implementation techniques in Rocq and Abella

Abella relies on

- unification (even of terms with binders) [Banff 1989]
- controlled backtracking search
- computation of functions presented as relations (NEW!).
- forward chaining and saturation (NEW!)

Rocq relies on

- rich type checking
- function programming style computation
- Separation of the kernel from proof refinement.
- Refinement is programmable using tactics

One commonality between Abella and Rocq

They both contain implementations of λProlog

- Abella supports the "two-level of logic approach". λProlog is the object-level specification.
- The Rocq-ELPI plug-in embeds the ELPI-λProlog into the Rocq prover. λProlog has access to Rocq structures such as proofs, specifications, logical expressions, etc.
  - Tassi et al., ELPI, Rocq-ELPI 2015-present.
  - "Trocq: Proof Transfer for Free, With or Without Univalence" by Cohen, Crance, and Mahboubi (ESOP 2024).

Conclusions

- Designing proof assistants based on proof theory should have its advantages.
- The proof theory treatment of binders in syntax
  - is natural and useful, and
  - is difficult to repeat in type theory based systems.
- Some future avenues:
  - Proof theory easily motivates richer terms structures, including a first-class treatment of sharing [M & Wu, CSL 2023].
  - Proof transformation, proof distillation, elaborating proof outlines. Logic programming technology allows for some proof-reconstruction during proof checking.
  - See: Matteo Manighetti's 2022 PhD "Developing proof theory for proof exchange".

Thanks

Questions?

Art by Nadia Miller