AI-assisted formalization of Π injectivity and uniqueness of typing for Lean's kernel

Mario Carneiro, Thierry Coquand, Meven Lennon-Bertrand
(with Claude as collaborator)

The metatheoretic load-bearing wall

Lean4Lean is a verified Lean 4 kernel: a re-implementation in Lean of the kernel, plus a formal specification of its abstract type theory and a proof that the implementation satisfies it. Most of the verification is tractable; the correctness argument is held up by a cluster of *mutually dependent* metatheorems:

- Π *injectivity*: $\Gamma \vdash \Pi A.B \equiv \Pi A'.B'$ implies $\Gamma \vdash A \equiv A'$ and $A :: \Gamma \vdash B \equiv B'$.
- *Sort injectivity*: $\Gamma \vdash \mathsf{sort}\, u \equiv \mathsf{sort}\, v$ implies $u \approx v$.
- *Uniqueness of typing*: $\Gamma \vdash e : A$ and $\Gamma \vdash e : B$ imply $\Gamma \vdash A \equiv B$.

These three prove each other and resist separation. The Rocq side has been wrestling with an essentially identical wall around η-conversion: in Lennon-Bertrand's WITS 2022 talk *À bas l'η*[1], the conversion-checking implementation is not the trouble — the metatheory is, since every plausible untyped formulation of η breaks confluence, subject reduction, or injectivity of type constructors. His *What Does It Take to Certify a Conversion Checker?* (FSCD 2025) gives the clean modular decomposition — injectivity of type constructors (positive soundness), term-level injectivities (negative soundness), normalisation (termination) — and his 2026 seminar talk *Injectivity of type constructors*[2] charts the available techniques, labelling the Coquand–Huber *domain model* approach we follow the "very unexplored" cell that handles η. The MetaRocq programme (Sozeau et al., *J. ACM* 2025) has scaled the rewriting/confluence cell to a realistic kernel but does not handle η. Filling in the domain-model cell for the full Lean type theory is the present work.

The Coquand–Huber escape

Coquand and Huber's *An Adequacy Theorem for Dependent Type Theory* (*Theory of Computing Systems* 63:647–665, 2019) was motivated by proving injectivity for Martin-Löf type theory in a *weak metatheory* — in particular, without strong normalization. The paper builds a domain model D and proves, by an "inclusive predicates" argument from classical domain theory, that two convertible terms have the same Böhm tree in D; injectivity of type constructors follows immediately.

The paper had been overlooked because the standard logical-relations machinery (e.g. Abel–Öhman–Vezzosi, POPL 2017) works under strong normalization. Lean spoils this: Abel and Coquand (*LMCS* 2020) gave an explicit counterexample using K-like reduction with Lean's impredicative `Prop`. Any proof for Lean must work without SN of the object theory — exactly Coquand–Huber's regime.

[1] `https://youtu.be/93E54Lra_Gc`
[2] `https://www.meven.ac/documents/26-tt-impl.pdf`

Type-in-Type — Martin-Löf's original 1967 system, rejected as inconsistent — is a natural toy setting for non-normalizing DTT: the universe sits inside itself, all universes collapse to one level, and the absence of SN is the default. We first worked the Coquand–Huber argument out there (`Lean4Lean/Experimental/Thierry.lean`), then scaled the construction up to the full Lean type theory: cumulative universes, pattern-style reductions, inductive families, propositional irrelevance.

A consequence is *robustness under theory variation.* Type-in-Type is a maximally strong theory (a side benefit of inconsistency), and the complications in mainstream DTTs — universe stratification, predicativity, impredicative-`Prop`/K, sized types — exist precisely to recover its strength while ruling out paradoxical constructions. These restrict which terms can be formed, not which model-theoretic identifications hold, so we expect the same construction to yield Π/sort injectivity for Rocq's and Agda's kernels too — a candidate *common core.*

AI-assisted formalization

The bulk of the Lean development took place over March–May 2026. Coquand produced an Agda transcription of the core proof in days using AI assistance; Lennon-Bertrand provided the Rocq-side perspective on the obstructions that Type-in-Type circumvents. The Agda codebase is ∼20 000 lines across 18 files, zero postulates, well commented; it is not unreadable, but the mathematical ideas are spread thin across files of selection/coherence/evaluation plumbing whose uniform naming does not cue which lemmas are load-bearing. The first author's Agda setup is also less ergonomic than for Lean.

What worked was using a frontier LLM (Claude, with the Agda and paper proof in context) as an *idea extractor*: it summarised Agda lemmas structurally, identified the analogous Lean construct, and proposed translations; the human checked each against the metatheoretic intent and had the model discharge the mechanics against the Lean LSP. A notable side effect: AI *compresses* formal artefacts — the Lean logical-relation file plus the adequacy proof come to ∼2000 lines, an order of magnitude below the Agda, even though Lean targets a richer object theory.

The human contributed architectural taste, strategic scope, and historical context (failed prior approaches, not in the codebase); the model contributed mechanical refactoring at scale and patient case analysis. Long context (∼1M tokens) let it carry the Agda, the paper, and several thousand lines of in-flight Lean simultaneously; the back-and-forth resembled sustained mathematical conversation, and triangulating across paper + Agda + human caught defects in either formal source. Work was *concurrent*: the first author edited in parallel with the model (frequent edit-fail collisions in the log), both cleaning up the model's bloat and *golfing* the proof back to something the human could sign off on — a deliberate counterweight to the LLM-bloat trajectory of the Agda source.

Status and ongoing work

The Lean development is nearly complete modulo one piece of stratification bookkeeping. Lean's `IsDefEq` transitivity is homogeneous in the type, but once type-equality is encoded as term-equality at a sort, the type-trans rule becomes heterogeneous — $A \equiv B : \mathsf{sort}\, u, B \equiv C : \mathsf{sort}\, v \implies A \equiv C : \mathsf{sort}\, u$ — and closing it requires identifying u with v, which is precisely the `sort_uniq` we are proving by well-founded induction. The Agda development avoids this entirely by splitting the judgment from the start into `TypeDefEq` and `TermDefEq`, homogeneous in each. We are exploring two repairs — (a) adding a further stratification index so the sort_uniq closure is internal to the relation; and (b) Lennon-Bertrand's suggestion to split the judgment as Agda does, removing the heterogeneity at the source. Either closes the proof.