AI-assisted formalization of typing for Lean's kernel

Mario Carneiro, Thierry Coquand, Meven Lennon-Bertrand, and Claude

Chalmers University of Technology

Outline

The load-bearing theorems

Lean4Lean

A *verified Lean 4 kernel*:

- Re-implementation of the kernel *in Lean*.
- Formal specification of the abstract type theory.
- Proof that the implementation satisfies the specification.

Most of the verification (WHNF, definitional unfolding, environment WF) is tractable.

The correctness argument hangs on three mutually dependent metatheorems.

The three theorems

- Π **injectivity**
  $\Gamma \vdash \Pi A.B \equiv \Pi A'.B' \implies \Gamma \vdash A \equiv A'$ and $A :: \Gamma \vdash B \equiv B'$.

The three theorems

- 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**
  $\Gamma \vdash \mathsf{sort}\, u \equiv \mathsf{sort}\, v \implies u \approx v$.

The three theorems

- 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 $\Gamma \vdash e : B \implies \Gamma \vdash A \equiv B$.

The three theorems

- 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 $\Gamma \vdash e : B \implies \Gamma \vdash A \equiv B$.

Each is used to prove the others. They resist separation.

The Rocq side has been here for a while

Siles & Herbelin, *JFP* 2012: typed and untyped conversion in Pure Type Systems are equivalent.

$\Rightarrow$ the kernel can do conversion *untyped* and still meet the typed-conversion spec.

The Rocq side has been here for a while

Siles & Herbelin, *JFP* 2012: typed and untyped conversion in Pure Type Systems are equivalent.
$\Rightarrow$ the kernel can do conversion *untyped* and still meet the typed-conversion spec.

Lennon-Bertrand, WITS 2022 — *À bas l'*η:

- The conversion-checking *implementation* is fine.
- The *metatheory* is what's hard.
- Every untyped formulation of η breaks something (confluence, SR, or injectivity).

The Rocq side has been here for a while

Siles & Herbelin, *JFP* 2012: typed and untyped conversion in Pure Type Systems are equivalent.
$\Rightarrow$ the kernel can do conversion *untyped* and still meet the typed-conversion spec.

Lennon-Bertrand, WITS 2022 — *À bas l'*η:

- The conversion-checking *implementation* is fine.
- The *metatheory* is what's hard.
- Every untyped formulation of η breaks something (confluence, SR, or injectivity).

Lennon-Bertrand, FSCD 2025 — modular decomposition:

injectivity of type constructors	$\rightarrow$ positive soundness
term-level injectivities	$\rightarrow$ negative soundness
normalisation	$\rightarrow$ termination

The technique landscape (Lennon-Bertrand 2026)

Four approaches to proving injectivity:

	Logical Relations	Rewriting /Confluence	Gluing	**Domain Model**
handles η	✓	×	✓	✓
weak ambient theory	×	✓	×	✓
gives normalisation	✓	×	✓	×
scales to real kernels	×	✓	?	?
state of the art	most explored	MetaRocq (*J.ACM* '25)	hard to formalise	very unexplored

We fill in the Domain Model cell — for full Lean.

Coquand & Huber 2019

An Adequacy Theorem for Dependent Type Theory, *Theory of Computing Systems* 63, 2019.

- Domain model D of dependent type theory.
- Inclusive-predicates argument $\Rightarrow$ two convertible terms have the same Böhm tree in D.
- $\Rightarrow$ Injectivity of type constructors falls out.
- Built for a *weak metatheory* — no SN assumed.

Why this matters now: Lean is not SN

Standard logical-relations machinery (Abel–Öhman–Vezzosi POPL 2017) works under strong normalization.

Lean's type theory *is not* strongly normalizing: Abel & Coquand (*LMCS* 2020) gave an explicit counterexample using K-like reduction with Lean's impredicative `Prop`.

Any proof of Π/sort injectivity for Lean must work without SN of the object theory — exactly the regime Coquand–Huber's construction addresses.

Type-in-Type as a natural toy setting

Martin-Löf 1967: $\mathsf{Type} : \mathsf{Type}$. Rejected as inconsistent, mostly avoided since.

For our purposes it is ideal:

- Universe sits inside itself.
- All universes collapse to one level.
- Absence of SN is the *default*, not a patch.

We worked the toy version out first in `Lean4Lean/Experimental/Thierry.lean`, then scaled up.

Robustness: scales to richer theories

Type-in-Type is *maximally strong* (inconsistent).

Mainstream DTTs' complications — universe stratification, predicativity, impredicative-`Prop`/K, sized types — exist to recover its strength while ruling out paradoxes.

These restrict which terms can be *formed*, not which model-theoretic identifications hold.

⇒ The adequacy proof scales to cumulative universes, pattern reductions, inductive families, proof-irrelevance — essentially unchanged.

Roadmap for the rest of the talk

Now I'll turn to what's specific to the Lean side of the joint work:

1. **The AI workflow**
   How Claude got from Thierry's Agda (20 000 lines) to working Lean (2 000 lines) over March–May 2026.

Roadmap for the rest of the talk

Now I'll turn to what's specific to the Lean side of the joint work:

1. **The AI workflow**
   How Claude got from Thierry's Agda (20 000 lines) to working Lean (2 000 lines) over March–May 2026.

2. **The collaboration shape**
   Concurrent human + LLM editing, cross-source triangulation, deliberate golfing.

Roadmap for the rest of the talk

Now I'll turn to what's specific to the Lean side of the joint work:

1. **The AI workflow**
   How Claude got from Thierry's Agda (20 000 lines) to working Lean (2 000 lines) over March–May 2026.

2. **The collaboration shape**
   Concurrent human + LLM editing, cross-source triangulation, deliberate golfing.

3. **A Lean-specific technical wrinkle**
   Heterogeneous transitivity — which the Agda development sidesteps by design, and Lean has to dispatch in its own way.

AI-assisted formalization

The Agda source (what I'm starting from)

- ~20,000 lines across 18 files.
- Zero postulates. Well commented. Cleanly structured.

The Agda source (what I'm starting from)

- ~20,000 lines across 18 files.
- Zero postulates. Well commented. Cleanly structured.

—*Claude*

The Agda source (what I'm starting from)

- ~20,000 lines across 18 files.
- Zero postulates. Well commented. Cleanly structured.

—*Claude*

- **Not unreadable — the difficulty is volume.**

Mathematical ideas spread thin across files of selection / coherence / evaluation plumbing. Names are uniform (`CoherentFunTail-append`, `EvalFun-mon-arg`, ...) so the syntax does not cue which lemmas are load-bearing.

(And my Agda setup is much less ergonomic than for Lean.)

LLM as idea extractor

Claude, ~1M-token context, with Agda + paper + in-flight Lean all in scope.

- Summarise each Agda lemma *structurally*.
- Identify the analogous Lean construct.
- Propose a Lean translation.

LLM as idea extractor

Claude, ~1M-token context, with Agda + paper + in-flight Lean all in scope.

- Summarise each Agda lemma *structurally*.
- Identify the analogous Lean construct.
- Propose a Lean translation.

Human passes:

- Check each proposal against metatheoretic intent.
- Have model discharge mechanics against the Lean LSP.

Side effect: AI compresses formal artefacts

20,000 lines of Agda $\longrightarrow$ 2,000 lines of Lean

Order of magnitude reduction — even though Lean targets a *richer* object theory.

Two files (`ShapeLogRel.lean` + `ShapeLogRelAdequacy.lean`) carry the core construction.

Side effect: AI compresses formal artefacts

20,000 lines of Agda $\longrightarrow$ 2,000 lines of Lean

Order of magnitude reduction — even though Lean targets a *richer* object theory.

Two files (`ShapeLogRel.lean` + `ShapeLogRelAdequacy.lean`) carry the core construction.

Not a claim that Lean is more concise than Agda.
A claim that LLM-mediated translation flattens incidental verbosity while leaving structure intact.

Concurrent workflow, not turn-taking

Human + model editing the same files in parallel.
Session log carries frequent edit-fail collisions.

Human contributed

- Architectural taste
- Strategic scope
- Historical context (failed approaches)

Model contributed

- Mechanical refactoring at scale
- Patient case analysis
- Long-context recall across thousands of lines

Downtime traded both ways: when I got tired I'd let Claude run; when Claude hit rate limits I'd think some more.

Human passes also *golfed* the proof down to a form the author could read and sign off on.
A deliberate counterweight to the LLM-bloat trajectory observed in the Agda source.

Cross-source triangulation

$$\text{Paper} \quad \leftrightarrow \quad \text{Agda} \quad \leftrightarrow \quad \text{Lean}$$

(human as referee)

Three independent synthesis sources. Defects in any one source were caught by redundancy, not by formal verification:

- Agda had a significant termination workaround that altered the proof structure.
- Translating from the paper *and* the Agda in parallel, Claude produced a Lean version that did *not* inherit the workaround.
- The cleaner of the two synthesis sources won out.

Status

Where we are

Lean development is complete on the architectural side:

- Π injectivity, sort injectivity, uniqueness of typing all stated and proved.
- The heterogeneous-transitivity problem (next slides) has been resolved.

Remaining sorries are in the `const` and `extra` (pattern-unfolding) cases of the adequacy induction — localised facts about constant application and δ-style rewriting, not structural to the LR construction.

The heterogeneous transitivity problem

Original rule:

$$\Gamma \vdash A \equiv B\,\mathsf{type},\ \Gamma \vdash B \equiv C\,\mathsf{type} \implies \Gamma \vdash A \equiv C\,\mathsf{type}$$

The heterogeneous transitivity problem

Original rule:

$$\Gamma \vdash A \equiv B\,\mathsf{type},\ \Gamma \vdash B \equiv C\,\mathsf{type} \implies \Gamma \vdash A \equiv C\,\mathsf{type}$$

Once type-equality is encoded as term-equality at a sort:

$$\Gamma \vdash A \equiv B : \mathsf{sort}\,u,\ \Gamma \vdash B \equiv C : \mathsf{sort}\,v \implies \Gamma \vdash A \equiv C : \mathsf{sort}\,u$$

u and v may differ. Lean's `IsDefEq` transitivity requires them to match.

The heterogeneous transitivity problem

Original rule:

$$\Gamma \vdash A \equiv B\,\mathsf{type},\ \Gamma \vdash B \equiv C\,\mathsf{type} \implies \Gamma \vdash A \equiv C\,\mathsf{type}$$

Once type-equality is encoded as term-equality at a sort:

$$\Gamma \vdash A \equiv B : \mathsf{sort}\,u,\ \Gamma \vdash B \equiv C : \mathsf{sort}\,v \implies \Gamma \vdash A \equiv C : \mathsf{sort}\,u$$

u and v may differ. Lean's `IsDefEq` transitivity requires them to match.

$\Rightarrow$ We need `sort_uniq` to identify $u \approx v$.
But `sort_uniq` *is what we are proving* by well-founded induction.

Forces IH appeals at various deep points of adequacy.

The Agda development sidesteps this entirely

Coquand made a structural choice from the start:

- Separate judgments for type-equality and term-equality.
- `TypeDefEq` and `TermDefEq`.
- Transitivity is *homogeneous* within each.
- No heterogeneous trans rule to close.

Resolution: extend the theory, then prove the extension redundant

Define $\mathsf{IsDefEq}'$ = IsDefEq + a new heterogeneous-trans constructor:

$$\Gamma \vdash A \equiv B : \mathsf{sort}\, u,\ \Gamma \vdash B \equiv C : \mathsf{sort}\, v \implies \Gamma \vdash A \equiv C : \mathsf{sort}\, u$$

Two-step proof:

1. Run the LR adequacy construction over $\mathsf{IsDefEq}'$ — the new constructor is now a *rule*, not a derived lemma, so there is no IH to chase.
2. Prove $\mathsf{IsDefEq}' \iff \mathsf{IsDefEq}$ — i.e. the new constructor is *admissible*.

Step 1 makes the adequacy proof structurally clean. Step 2 then transfers all the LR consequences back to vanilla IsDefEq.

Several earlier candidate fixes (extra stratification index; splitting the judgment) turned out to be unnecessary. The exploratory phase — where the LLM was useful for prototyping failed designs to expose structural problems — steered us to this simpler "extend and then collapse" resolution.

Conclusion

What we did:

- Filled in the Coquand–Huber domain-model cell of the injectivity-technique landscape for the full Lean type theory.
- Dispatched the heterogeneous-transitivity wrinkle by extending IsDefEq with the rule and then proving it admissible.
- Used LLM-mediated translation to compress 20k lines of Agda into 2k lines of Lean — with concurrent human golfing to preserve the quality standards.

Conclusion

What we did:

- Filled in the Coquand–Huber domain-model cell of the injectivity-technique landscape for the full Lean type theory.
- Dispatched the heterogeneous-transitivity wrinkle by extending IsDefEq with the rule and then proving it admissible.
- Used LLM-mediated translation to compress 20k lines of Agda into 2k lines of Lean — with concurrent human golfing to preserve the quality standards.

What's next:

- Close the remaining `const` and `extra` (pattern-unfolding) sorries — localised, no architectural risk.
- Wire the resulting injectivity / uniqueness theorems back into the Lean4Lean kernel correctness statement: the load-bearing wall comes down.
- Port the same construction to Rocq and Agda kernels — the domain-model argument is largely theory-agnostic, and the candidate *common core* is the main long-term bet.

Thank you!

Questions?

https://github.com/digama0/lean4lean/blob/master/Lean4Lean/Experimental/ShapeLogRelAdequacy.lean