Lean4Lean:
Mechanizing the metatheory of Lean

Mario Carneiro

Chalmers University of Technology, University of Gothenburg

June 13, 2025

Introduction

Introduction

- I don't think I need to explain what Lean is to this crowd

Introduction

- I don’t think I need to explain what Lean is to this crowd
- What will be relevant for this talk:
  - Lean is an ITP (Interactive Theorem Prover)

Introduction

- I don't think I need to explain what Lean is to this crowd
- What will be relevant for this talk:
  - Lean is an ITP (Interactive Theorem Prover)
  - It is based on Dependent Type Theory

Introduction

- I don't think I need to explain what Lean is to this crowd
- What will be relevant for this talk:
  - Lean is an ITP (Interactive Theorem Prover)
  - It is based on Dependent Type Theory
    - Specifically, an extension of Martin-Löf Type Theory (MLTT) called the Calculus of Inductive Constructions (CIC)

Lean4Lean

- In 2019 I worked out the properties of LeanTT for my masters thesis

Lean4Lean

- In 2019 I worked out the properties of LeanTT for my masters thesis
- It has been 5 years since then and we have a new Lean version now, which made some kernel changes

Lean4Lean

- In 2019 I worked out the properties of LeanTT for my masters thesis
- It has been 5 years since then and we have a new Lean version now, which made some kernel changes
- Lean 4 has had a few (minor, short-lived) soundness bugs

Lean4Lean

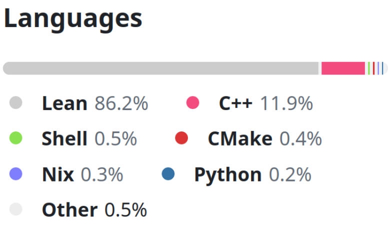
- In 2019 I worked out the properties of LeanTT for my masters thesis
- It has been 5 years since then and we have a new Lean version now, which made some kernel changes
- Lean 4 has had a few (minor, short-lived) soundness bugs
- So I've been working on a project to formalize the kernel

Lean4Lean

- In 2019 I worked out the properties of LeanTT for my masters thesis
- It has been 5 years since then and we have a new Lean version now, which made some kernel changes
- Lean 4 has had a few (minor, short-lived) soundness bugs
- So I've been working on a project to formalize the kernel
  - One sentence summary: "It's MetaRocq, but for Lean"

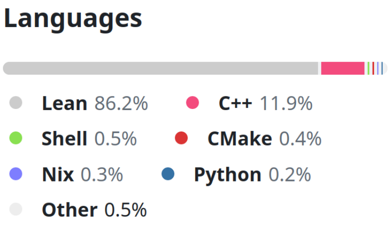
Bootstrapping Lean

- Lean is about 80% written in Lean, including:
  - The parser
  - The elaborator
  - The tactic language
  - The metaprogramming framework
  - The LSP server



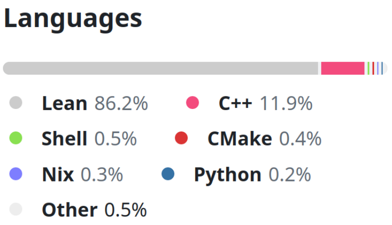
Bootstrapping Lean

- Lean is about 80% written in Lean, including:
  - The parser
  - The elaborator
  - The tactic language
  - The metaprogramming framework
  - The LSP server
- The exceptions are:
  - The runtime (very small)
  - The interpreter
  - Half of the backend of the old compiler
  - The kernel



Bootstrapping Lean

- Lean is about 80% written in Lean, including:
  - The parser
  - The elaborator
  - The tactic language
  - The metaprogramming framework
  - The LSP server
- The exceptions are:
  - The runtime (very small)
  - The interpreter
  - Half of the backend of the old compiler
  - **The kernel**
- Of these, one of them is both mathematically interesting and soundness critical...



Lean4Lean

Project goals:

1.
   - Make a Lean kernel...
   - which is *complete* for everything the original can handle
   - and competitive with the original so that it can be considered as a replacement.
2.
   - Write down the type theory of Lean (but formally, in Lean itself)
   - Prove structural properties about the type system
3.
   - Prove the correctness of the implementation with respect to the specification

Lean4Lean

Project goals:

1. ✓ Make a Lean kernel...
   ✓ which is *complete* for everything the original can handle
   ✓ and competitive with the original so that it can be considered as a replacement.
2. ✓ Write down the type theory of Lean (but formally, in Lean itself)
   ▲ Prove structural properties about the type system
3. ▲ Prove the correctness of the implementation with respect to the specification

The Lean4Lean kernel

- A carbon copy of the C++ code
- It does all the same fancy tricks as the original, and no more
  - ✓ Union-find data structures for caching
  - ✓ Pointer equality testing
  - ✓ Bidirectional typechecking
  - ✓ Identical def.eq. heuristics
  - ✓ η for structures, nested inductive types
  - × Naive implementation of substitution and reduction
- Suitable for differential fuzzing (e.g. it will get exactly the same counts for definition unfolding etc.)
- Uses Lean's own `Expr` type
  - A few algorithms are reused when they were already available in Lean

The Lean4Lean kernel

	`lean4export`	`lean4lean`	ratio
`Lean`	37.01 s	44.61 s	1.21
~~`Std`~~ `Batteries`	32.49 s	45.74 s	1.40
`Mathlib` (+ `Std` + `Lean`)	44.54 min	58.79 min	1.32

- Performance is about 30% worse than the original
  (How good this is depends on your temperament)
- Lean itself took hits of a similar order of magnitude when moving the elaborator out of C++, and that was worth it for the improved extensibility and development features
- It has since clawed back all the performance and then some by implementing better algorithms that were difficult to get right in C++
- I want to experiment with better reduction strategies, this is never going to happen with the current kernel

Recall: DTT

Dependent Type Theory

$$e ::= x \mid c \mid e\ e \mid \lambda x : e.\ e \mid \forall x : e.\ e \mid U_n$$
$$\Gamma ::= \cdot \mid \Gamma, x : e$$

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma \vdash e_1 : \forall x : \alpha.\ \beta \quad \Gamma \vdash e_2 : \alpha}{\Gamma \vdash e_1\ e_2 : \beta[e_2/x]}$$

$$\frac{\Gamma, x : \alpha \vdash e : \beta}{\Gamma \vdash (\lambda x : \alpha.\ e) : \forall x : \alpha.\ \beta} \qquad \frac{}{\Gamma \vdash U_n : U_{n+1}}$$

$$\frac{\Gamma \vdash \alpha : U_m \quad \Gamma, x : \alpha \vdash \beta : U_n}{\Gamma \vdash \forall x : \alpha.\ \beta : U_{\mathrm{imax}(m,n)}} \qquad \frac{\Gamma \vdash e : \alpha \quad \Gamma \vdash \alpha \equiv \beta}{\Gamma \vdash e : \beta}$$

Buzzword bingo

Lean's specific flavor of DTT has:

- an impredicative universe `Prop` of propositions
- algebraic universes (with max & imax,
  where $\mathrm{imax}(a, b) \ := \ \text{if } b = 0 \text{ then } 0 \text{ else } \max(a, b)$)
- definitional proof irrelevance
- no universe cumulativity
- indexed, mutual and nested inductive types
- an η reduction rule for lambdas and structures

Proof Irrelevance and its consequences

- We want to treat all proofs of a proposition as "the same"

$$\frac{\Gamma \vdash p : \mathbb{P} \quad \Gamma \vdash h : p \quad \Gamma \vdash h' : p}{\Gamma \vdash h \equiv h'}$$

- This means that an equality has at most one proof (anti-HoTT)
- To prevent inconsistency, some inductive types cannot eliminate to a large universe

$$\exists x : \alpha.\ p\ x := \mu T : \mathbb{P}.\ (\text{intro} : \forall x : \alpha.\ p\ x \to T)$$

$$\text{intro} : \forall x : \alpha.\ (p\ x \to \exists y : \alpha.\ p\ y)$$

$$\text{rec}_\exists : \forall C : U_0.(\forall x : \alpha.\ p\ x \to C) \to (\exists x : \alpha.\ p\ x) \to C$$

- Some inductive types in $\mathbb{P}$ eliminate to other universes, if they have "at most one inhabitant by definition", this is called **subsingleton elimination**

Actual axioms

- Propositional extensionality

$$\text{propext} : \forall p, q : \mathbb{P}.\ (p \leftrightarrow q) \to p = q$$

- Quotient types

$$\begin{aligned}
\text{quot} &: \forall \alpha : U_n.\ (\alpha \to \alpha \to \mathbb{P}) \to U_n \\
\text{mk}_{\alpha,r} &: \alpha \to \alpha/r \\
\text{lift}_{\alpha,r} &: \forall \beta.\ \forall f : \alpha \to \beta.\ (\forall x\, y.\ r\ x\ y \to f\ x = f\ y) \to \alpha/r \to \beta \\
\text{sound}_{\alpha,r} &: \forall x\, y.\ r\ x\ y \to \text{mk}\ x = \text{mk}\ y
\end{aligned}$$

$$\text{lift}\ \beta\ f\ H\ (\text{mk}\ x) \equiv f\ x$$

- The axiom of choice

$$\begin{aligned}
\text{nonempty}\ \alpha &:= \mu T : U_0.\ (\text{intro} : \alpha \to T) \\
\text{choice} &: \forall \alpha : U_n.\ \text{nonempty}\ \alpha \to \alpha
\end{aligned}$$

DTT in ZFC

- There is an "obvious" model of DTT in ZFC, where we treat types as sets and elements as elements of the sets
- The interpretation function $[\![\Gamma \vdash e]\!]_\gamma$ (or just $[\![e]\!]$) translates e into a set when $\Gamma \vdash e : \alpha$ is well typed and $\gamma \in [\![\Gamma]\!]$ provides a values for the context
- Because of proof irrelevance and the axiom of choice (which implies LEM), we must have $[\![\mathbb{P}]\!] := \{\emptyset, \{\bullet\}\}$
- For all higher universes, we interpret functions as functions, i.e. $f \in [\![\forall x : \alpha.\ \beta]\!]$ if f is a function with domain $[\![\alpha]\!]$ such that $f(x) \in [\![\beta]\!]_x$ for all $x \in [\![\alpha]\!]$
- With this translation, because of inductive types the universes must be very large (Grothendieck universes). We let $[\![U_{n+1}]\!] = V_{\kappa_n}$ where κ_n is the n-th inaccessible cardinal (if it exists)

Lean is consistent

Theorem (Soundness)

1. *If* $\Gamma \vdash \alpha : \mathbb{P}$*, then* $[\![\Gamma \vdash \alpha]\!]_\gamma \subseteq \{\bullet\}$
2. *If* $\Gamma \vdash e : \alpha$ *and* $\mathrm{lvl}(\Gamma \vdash \alpha) = 0$*, then* $[\![\Gamma \vdash e]\!]_\gamma = \bullet$.
3. *If* $\Gamma \vdash e : \alpha$*, then there exists* $k \in \mathbb{N}$ *such that if there are* k *inaccessible cardinals, then* $[\![\Gamma \vdash e]\!]_\gamma \in [\![\Gamma \vdash \alpha]\!]_\gamma$ *for all* $\gamma \in [\![\Gamma]\!]$.
4. *If* $\Gamma \vdash e \equiv e'$*, then there exists* $k \in \mathbb{N}$ *such that if there are* k *inaccessible cardinals, then* $[\![\Gamma \vdash e]\!]_\gamma = [\![\Gamma \vdash e']\!]_\gamma$ *for all* $\gamma \in [\![\Gamma]\!]$.

- As a consequence, Lean is consistent (there is no derivation of $\bot$), if ZFC with ω inaccessibles is consistent.
- More precisely, Lean is equiconsistent with ZFC + $\{$there are n inaccessibles $\mid n < \omega\}$, because Lean models ZFC + n inaccessibles for all $n < \omega$

Undecidability

- The type judgment is “almost” decidable, but not quite
- The problem is an interaction of subsingleton elimination (for Acc) and proof irrelevance
- ask me if you want more details

Algorithmic typing judgment

- Lean resolves this by underapproximating the $\equiv$ and $\vdash$ judgments
- If we introduce $\Gamma \vdash e \Leftrightarrow e'$ and $\Gamma \Vdash e : \alpha$ judgments for "the thing Lean does", then $\Gamma \Vdash e : \alpha$ implies $\Gamma \vdash e : \alpha$ and $\Gamma \vdash e \Leftrightarrow e'$ implies $\Gamma \vdash e \equiv e'$, so Lean is an **underapproximation** of the "true" typing judgment
  - We will not attempt to prove completeness of the kernel
- $\Gamma \vdash e \Leftrightarrow e'$ is not transitive, and $\Gamma \Vdash e : \alpha$ does not satisfy subject reduction
- $\Gamma \vdash e \equiv e'$ and $\Gamma \vdash e : \alpha$ are better behaved (by fiat), but undecidable
- In Lean4Lean we mostly concern ourselves with the abstract judgment

Unique typing: not yet a theorem

Conjecture (Unique typing)

If $\Gamma \vdash e : \alpha$ *and* $\Gamma \vdash e : \beta$, *then* $\Gamma \vdash \alpha \equiv \beta$.

Conjecture (Definitional inversion)

- *If* $\Gamma \vdash U_m \equiv U_n$, *then* $m = n$.
- *If* $\Gamma \vdash \forall x : \alpha.\ \beta \equiv \forall x : \alpha'.\ \beta'$, *then* $\Gamma \vdash \alpha \equiv \alpha'$ *and* $\Gamma, x : \alpha \vdash \beta \equiv \beta'$.
- *If* $\Gamma \vdash U_n \not\equiv \forall x : \alpha.\ \beta$.

- When formalizing this proof from my thesis, I found a gap in the proof
- I still believe the theorems are true
- There is an alternative path to the proof of soundness, but some of the kernel optimizations depend on this theorem

The Church Rosser theorem

Theorem (for Lean)

If $\Gamma \vdash e : \alpha$, and $\Gamma \vdash e \rightsquigarrow_\kappa^* e_1, e_2$, then there exists e_1', e_2' such that $\Gamma \vdash e_i \rightsquigarrow_\kappa^* e_i'$ and $\Gamma \vdash e_1' \equiv_p e_2'$.

- The statement uses two new relations, the κ reduction $\rightsquigarrow_\kappa$ and proof equivalence $\equiv_p$.
- $\rightsquigarrow_\kappa$ is a more aggressive version of Lean's reduction relation that unfolds subsingleton eliminators even on variables
- $\equiv_p$ is "equality except at proof arguments" with η conversion.

$$\frac{\Gamma \vdash e : \alpha}{\Gamma \vdash e \equiv_p e} \qquad \frac{\Gamma, x : \alpha \vdash e \equiv_p e'\ x}{\Gamma \vdash \lambda x : \alpha.\ e \equiv_p e'} \qquad \frac{\Gamma \vdash p : \mathbb{P} \quad \Gamma \vdash h, h' : p}{\Gamma \vdash h \equiv_p h'} \qquad \ldots$$

The Church Rosser theorem

- The $\rightsquigarrow_\kappa$ reduction will reduce $\mathrm{rec}_{\mathrm{acc}}\ C\, f\ x\, h$ (where $h : \mathrm{acc}_<\ x$) to

$$f\ x\ (\mathrm{inv}_x\ h)\ (\lambda y\, h'.\ \mathrm{rec}_{\mathrm{acc}}\ C\, f\ y\ (\mathrm{inv}_x\ h\ y\ h'))$$

  so it is not strongly or weakly normalizing
- So it is similar to the untyped lambda reduction in that by allowing infinite reduction we open the possibility of bringing divergent reductions back together (within $\equiv_p$)
- The proof of Church-Rosser as stated uses the Tait–Martin-Löf method (using a parallel reduction relation $\gg_\kappa$ and its almost deterministic analogue $\ggg_\kappa$)

Unique typing

- The proof used a stratification of the typing judgment for the induction order, but $\Gamma \vdash_n e : \alpha$ is not closed under substitution.
- The Church-Rosser part of the proof seems okay, but we need a smarter induction measure.
- For part 3 of the project, I have decided to set this proof aside and `sorry` it. The Lean kernel really depends on this property for correctness.
- Type theorists wanted!
  - I don't think this is a crazy impossible problem, but I'm doing too many projects at once (see: rest of the talk). I would be very happy if someone picked this up

Verifying the kernel

Verifying the kernel

Lean doesn't just implement the type theory as-is. It has a laundry list of optimizations over the obvious definition in almost every function. As a result, the verification is not at all straightforward.

- The theory uses a type `VExpr` while Lean uses `Expr`.
  - `Expr` uses "locally nameless" representation, while `VExpr` uses pure de Bruijn variables
  - `Expr` has additional primitives:
    - natural number and string literals
    - metavariables and free variables
    - primitive projections
  - `Expr` caches metadata like "do I have a free variable" inside every subexpression
- Some functions on natural numbers are overridden with a native implementation using GMP (GNU Multiple Precision arithmetic library)

Progress

- There has been some recent (~3 weeks) progress on the program verification part
- Most theorems stated in terms of a simple one-sided Hoare logic $s \models (x : M\ \alpha)\ \{Q\}$

```
theorem checkType.WF {c : VContext} {s : VState}
    (h1 : e.FVarsIn s.ngen.Reserves) :
    RecM.WF c s (inferType e false) fun ty _ => ∃ e' ty',
      c.TrExprS e e' ∧ c.TrExprS ty ty' ∧ c.HasType e' ty'
```

Verification pays off

- Just last week, I was working on proving `Expr.hasBoundVar` is correct
- This function works by accessing a 64 bit metadata field
- I had to prove some bit tricks correct, but this worked out alright
- But the metadata calculates the depth of bound variables, which is a priori unbounded, so this is an overflow situation
- The code was checking for overflow, but it used Lean's `panic` function to do it, and this function doesn't actually crash the program, because it's a pure function with no exit path
- Instead, it returned the default value for the type (0), which is the worst possible answer in this situation
- By pushing back the counterexample situation to the entry point I was able to construct a concrete proof of false [#8554].

Summary

- You can use Lean4Lean as a replacement for Lean's kernel today
- The formalization is still under active development, not all mathematical problems are solved yet
- There are a half dozen people working on MetaRocq, but Lean doesn't have enough type theorists involved. If you identify as such, come help out!

`https://github.com/digama0/lean4lean`