Lean4Lean:

Formalizing the metatheory of Lean

Mario Carneiro

Carnegie Mellon University

July 24, 2024

Lean

- An open source interactive theorem prover developed primarily by Leonardo de Moura (Microsoft Research)
- Focus on software verification and formalized mathematics
- Based on **Dependent Type Theory**
  - Classical, non-HoTT
  - Similar to CIC, the axiom system used by Coq
- Lean 3 includes a powerful metaprogramming infrastructure for Lean in Lean
- The mathlib library for Lean 3 provides a broad range of pure mathematics and tools for (meta)programming

Axioms of Lean

Untyped Lambda Calculus

$$e ::= x \mid e\ e \mid \lambda x.\ e$$

$$\frac{e_1 \rightsquigarrow e_1'}{e_1\ e_2 \rightsquigarrow e_1'\ e_2} \qquad \frac{e_2 \rightsquigarrow e_2'}{e_1\ e_2 \rightsquigarrow e_1\ e_2'} \qquad \frac{}{(\lambda x.\ e')\ e \rightsquigarrow e'[e/x]}$$

- Originally developed by Alonzo Church as a simple model of computation (equivalent to Turing Machines)
- Primitive notion of bound variables and substitution
- Nondeterministic "reduction" operation on terms simulates execution
- Reduction is confluent (Church-Rosser theorem): If $e \rightsquigarrow^* e_1$ and $e \rightsquigarrow^* e_2$, then there exists e' such that $e_1, e_2 \rightsquigarrow^* e'$

Simple Type Theory

$$\tau ::= \iota \mid \tau \to \tau$$

$$e ::= x \mid e\ e \mid \lambda x \colon \tau.\ e$$

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma \vdash e_1 : \alpha \to \beta \quad \Gamma \vdash e_2 : \alpha}{\Gamma \vdash e_1\ e_2 : \beta} \qquad \frac{\Gamma, x : \alpha \vdash e : \beta}{\Gamma \vdash (\lambda x : \alpha.\ e) : \alpha \to \beta}$$

- Also developed by Alonzo Church as a type system over the untyped lambda calculus
- All terms normalize in this calculus (strong normalization)

Dependent Type Theory

$$\tau ::= \iota \mid \tau \to \tau$$

$$e ::= x \mid e\ e \mid \lambda x : \tau.\ e$$

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma \vdash e_1 : \alpha \to \beta \quad \Gamma \vdash e_2 : \alpha}{\Gamma \vdash e_1\ e_2 : \beta}$$

$$\frac{\Gamma, x : \alpha \vdash e : \beta}{\Gamma \vdash (\lambda x : \alpha.\ e) : \alpha \to \beta}$$

Dependent Type Theory

$$\tau ::= \iota \mid \forall x : \tau.\ \tau$$

$$e ::= x \mid e\ e \mid \lambda x : \tau.\ e$$

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

$$\frac{\Gamma, x : \alpha \vdash e : \beta}{\Gamma \vdash (\lambda x : \alpha.\ e) : \forall x : \alpha.\ \beta}$$

Dependent Type Theory

$$\tau ::= \iota \mid \forall x : \tau.\ \tau \mid U$$

$$e ::= x \mid e\ e \mid \lambda x : \tau.\ e$$

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

$$\frac{\Gamma, x : \alpha \vdash e : \beta}{\Gamma \vdash (\lambda x : \alpha.\ e) : \forall x : \alpha.\ \beta}$$

$$\frac{}{\Gamma \vdash \iota : U} \qquad \frac{\Gamma \vdash \alpha : U \quad \Gamma, x : \alpha \vdash \beta : U}{\Gamma \vdash \forall x : \alpha.\ \beta : U} \qquad \frac{}{\Gamma \vdash U : U}$$

Dependent Type Theory

$$e ::= x \mid e\ e \mid \lambda x : e.\ e \mid \iota \mid \forall x : e.\ e \mid U$$

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

$$\frac{\Gamma, x : \alpha \vdash e : \beta}{\Gamma \vdash (\lambda x : \alpha.\ e) : \forall x : \alpha.\ \beta}$$

$$\frac{}{\Gamma \vdash \iota : U} \qquad \frac{\Gamma \vdash \alpha : U \quad \Gamma, x : \alpha \vdash \beta : U}{\Gamma \vdash \forall x : \alpha.\ \beta : U} \qquad \frac{}{\Gamma \vdash U : U}$$

Dependent Type Theory

$$e ::= x \mid e\ e \mid \lambda x : e.\ e \mid \forall x : e.\ e \mid U$$

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

$$\frac{\Gamma, x : \alpha \vdash e : \beta}{\Gamma \vdash (\lambda x : \alpha.\ e) : \forall x : \alpha.\ \beta}$$

$$\frac{\Gamma \vdash \alpha : U \quad \Gamma, x : \alpha \vdash \beta : U}{\Gamma \vdash \forall x : \alpha.\ \beta : U} \qquad \frac{}{\Gamma \vdash U : U}$$

Two Problems

$$\frac{}{\Gamma \vdash U : U}$$

- Girard's paradox: This rule causes an inconsistency (all types become nonempty, i.e. all propositions are provable)
- Solution: hierarchies of universes

$$\frac{}{\Gamma \vdash U_{n} : U_{n+1}} \qquad \frac{\Gamma \vdash \alpha : U_{m} \quad \Gamma, x : \alpha \vdash \beta : U_{n}}{\Gamma \vdash \forall x : \alpha .\, \beta : U_{\max(m,n)}}$$

Impredicativity

- Curry-Howard correspondence: Propositions act like types, whose terms are the proofs ($\to$ and $\forall$ act like the logical operators $\to$ and $\forall$)
- We identify the lowest universe $\mathbb{P} := U_0$ as the universe of propositions
- We want things like "all natural numbers are even or odd" to be propositions, but the $\forall$ rule doesn't give us this

$$\frac{\Gamma \vdash \mathbb{N} : U_1 \quad \Gamma, n : \mathbb{N} \vdash \text{even } n \lor \text{odd } n : U_0}{\Gamma \vdash \forall n : \mathbb{N}.\ \text{even } n \lor \text{odd } n : U_1}$$

- Solution: fix the rule so that if the second argument is in U_0 then so is the forall

$$\frac{\Gamma \vdash \alpha : U_m \quad \Gamma, x : \alpha \vdash \beta : U_n}{\Gamma \vdash \forall x : \alpha.\ \beta : U_{\mathrm{imax}(m,n)}} \qquad \mathrm{imax}(m,n) = \begin{cases} 0 & n = 0 \\ \max(m,n) & \text{otherwise} \end{cases}$$

Dependent Type Theory

$$e ::= x \mid e\ e \mid \lambda x : e.\ e \mid \forall x : e.\ e \mid U_n$$

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

$$\frac{\Gamma, x : \alpha \vdash e : \beta}{\Gamma \vdash (\lambda x : \alpha.\ e) : \forall x : \alpha.\ \beta}$$

$$\frac{}{\Gamma \vdash U_n : U_{n+1}} \qquad \frac{\Gamma \vdash \alpha : U_m \quad \Gamma, x : \alpha \vdash \beta : U_n}{\Gamma \vdash \forall x : \alpha.\ \beta : U_{\mathrm{imax}(m,n)}}$$

Two Problems

- The types $(\lambda \alpha : U_1.\ \alpha)\ \tau$ and τ are not the same, even though $(\lambda \alpha : U_1.\ \alpha)\ \tau \rightsquigarrow \tau$
- Solution: Convertibility (a.k.a. definitional equality)

$$\frac{\Gamma \vdash e : \alpha \quad \Gamma \vdash \alpha \equiv \beta}{\Gamma \vdash e : \beta}$$

Two Problems

- The types $(\lambda\alpha : U_1.\ \alpha)\ \tau$ and τ are not the same, even though $(\lambda\alpha : U_1.\ \alpha)\ \tau \rightsquigarrow \tau$
- Solution: Convertibility (a.k.a. definitional equality)

$$\frac{\Gamma \vdash e : \alpha \quad \Gamma \vdash \alpha \equiv \beta}{\Gamma \vdash e : \beta}$$

$$\frac{\Gamma \vdash e : \alpha}{\Gamma \vdash e \equiv e} \qquad \frac{\Gamma \vdash e \equiv e'}{\Gamma \vdash e' \equiv e} \qquad \frac{\Gamma \vdash e_1 \equiv e_2 \quad \Gamma \vdash e_2 \equiv e_3}{\Gamma \vdash e_1 \equiv e_3}$$

$$\frac{\Gamma \vdash e_1 \equiv e_1' \qquad \Gamma \vdash e_2 \equiv e_2'}{\Gamma \vdash e_1\ e_2 \equiv e_1'\ e_2'} \qquad \frac{\Gamma \vdash \alpha \equiv \alpha' \quad \Gamma, x : \alpha \vdash \beta \equiv \beta'}{\begin{array}{c}\Gamma \vdash \lambda x : \alpha.\ \beta \equiv \lambda x : \alpha'.\ \beta' \\ \Gamma \vdash \forall x : \alpha.\ \beta \equiv \forall x : \alpha'.\ \beta'\end{array}}$$

$$\frac{}{\Gamma \vdash (\lambda x : \alpha.\ e')\ e \equiv e'[e/x]} \qquad \frac{\Gamma \vdash e : \beta}{\Gamma \vdash \lambda x : \alpha,\ e\ x \equiv e}$$

Inductive Types

- We want a general framework for defining new inductive types like $\mathbb{N}$

$$K ::= 0 \mid (c : e) + K$$

$$e ::= \cdots \mid \mu x : e.\ K \mid c_{\mu x : e . K} \mid \mathrm{rec}_{\mu x : e . K}$$

$$\begin{aligned}
\mathbb{N} &:= \mu T : U_1.\ (\mathrm{zero} : T) + (\mathrm{succ} : T \to T) \\
\exists x : \alpha.\ p\ x &:= \mu T : \mathbb{P}.\ (\mathrm{intro} : \forall x : \alpha.\ p\ x \to T) \\
p \wedge q &:= \mu T : \mathbb{P}.\ (\mathrm{intro} : p \to q \to T) \\
p \vee q &:= \mu T : \mathbb{P}.\ (\mathrm{inl} : p \to T) + (\mathrm{inr} : q \to T) \\
\bot &:= \mu T : \mathbb{P}.\ 0 \\
\top &:= \mu T : \mathbb{P}.\ (\mathrm{trivial} : T)
\end{aligned}$$

Inductive Types

- Each inductive type comes with a **constructor** for each case, and a **recursor** that allows us to prove theorems by induction and construct functions by recursion

$$
\begin{aligned}
\mathbb{N} &:= \mu T : U_1.\ (\mathrm{zero} : T) + (\mathrm{succ} : T \to T) \\
\mathrm{zero} &: \mathbb{N} \\
\mathrm{succ} &: \mathbb{N} \to \mathbb{N} \\
\mathrm{rec}_{\mathbb{N}} &: \forall (C : \mathbb{N} \to U_i).\ C\ \mathrm{zero} \to \\
&\qquad (\forall n : \mathbb{N}.\ C\ n \to C\ (\mathrm{succ}\ n)) \to \forall n : \mathbb{N}.\ C\ n
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{rec}_{\mathbb{N}}\ C\ z\ s\ \mathrm{zero} &\equiv z \\
\mathrm{rec}_{\mathbb{N}}\ C\ z\ s\ (\mathrm{succ}\ n) &\equiv s\ n\ (\mathrm{rec}_{\mathbb{N}}\ C\ z\ s\ n)
\end{aligned}
$$

Inductive Types

- For an inductive declaration to be admissible, it must be strictly positive (no T appears left of left of an arrow)
  - Ex: this type violates Cantor's theorem

$$\mathrm{bad} := \mu T : U_1.\ (\mathrm{intro} : (T \to 2) \to T)$$

- Inductive families are also allowed:

$$\begin{aligned}
\mathrm{eq}_\alpha &:= \lambda x : \alpha.\ \mu T : \alpha \to \mathbb{P}.\ (\mathrm{refl} : T\ x) \\
\mathrm{refl}_x &: \mathrm{eq}_\alpha\ x\ x \\
\mathrm{rec}_{\mathrm{eq}\ x} &: \forall (C : \alpha \to U_i).\ C\ x \to \forall y : \alpha.\ \mathrm{eq}_\alpha\ x\ y \to C\ y
\end{aligned}$$

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

$$\exists x : \alpha.\ p\ x := \mu T : \mathbb{P}.\ (\text{intro} : \forall x : \alpha.\ p\ x \to T)$$
$$\text{intro} : \forall x : \alpha.\ (p\ x \to \exists y : \alpha.\ p\ y)$$
$$\text{rec}_{\exists} : \forall C : U_{\color{red}{0}}.(\forall x : \alpha.\ p\ x \to C) \to (\exists x : \alpha.\ p\ x) \to C$$

- 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

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

- Quotient types

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

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

- The axiom of choice

$$\mathrm{nonempty}\ \alpha := \mu T : U_0.\ (\mathrm{intro} : \alpha)$$
$$\mathrm{choice} : \forall \alpha : U_n.\ \mathrm{nonempty}\ \alpha \to \alpha$$

Properties of the type system

Undecidability

- The type judgment is "almost" decidable, but not quite
- The problem is an interaction of subsingleton elimination and proof irrelevance

$$\mathrm{acc}_{<} := \mu T : \alpha \to \mathbb{P}.\ (\mathrm{intro} : \forall x.\ (\forall y.\ y < x \to T\ y) \to T\ x)$$

- acc x expresses that x is "accessible" via the $<$ relation
  - If everything $<$-less than x is accessible, then x is accessible
  - If everything is $<$-accessible then $<$ is a well founded relation
  - acc is a subsingleton eliminator that lives in $\mathbb{P}$!
- We can define an inverse to intro, such that intro x $(\mathrm{inv}_x\ a) \equiv a$, because a : acc x is a proposition

$$\mathrm{inv}_x : \mathrm{acc}\ x \to \forall y.\ y < \mathrm{acc}\ x \to \mathrm{acc}\ y$$

Undecidability

- Let P be a decidable proposition such that $\forall n.\ P\ n$ is not decidable
  - for example, $P\ n :=$ Turing machine M runs for at least n steps
- Suppose $h_0 : \mathrm{acc}_{>}\ 0$, that is, $0 : \mathbb{N}$ is accessible via the $>$ relation. (This is provably false.)
- We can define a function $f : \forall x : \mathbb{N}.\ \mathrm{acc}_{>}\ x \to \mathbb{N}$ by recursion on $\mathrm{acc}_{>}$ such that

  $$f\ n\ h \equiv \text{if } P\ n \text{ then } f\ (n+1)\ (\mathrm{inv}_n\ h\ (n+1)\ (p\ n)) \text{ else } 0$$

  where $p\ n : n + 1 > n$
- Then $h_0 : \mathrm{acc}_{>}\ 0 \vdash f\ 0\ h_0 \equiv 0$ is provable iff $\forall n.\ P\ n$ is true

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
- $\Gamma \vdash e \Leftrightarrow e'$ is not transitive, and $\Gamma \Vdash e : \alpha$ does not satisfy subject reduction
- In practice, this issue is extremely rare and it can be circumvented by inserting identity functions to help Lean find the transitivity path

Modeling Lean in ZFC

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 $\perp$), 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 $n < \omega$

Unique typing

Unique typing

- We used the function $\mathrm{lvl}(\Gamma \vdash \alpha)$ in the soundness theorem. This is defined as $\mathrm{lvl}(\Gamma \vdash \alpha) = n$ iff $\Gamma \vdash \alpha : U_n$, and it is well defined on types because of unique typing and definitional inversion:

Theorem (Unique typing)

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

Theorem (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 $\Gamma \vdash U_n \not\equiv \forall x : \alpha.\, \beta$.*

Unique typing

Theorem (Unique typing)

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

- Note that this works even in inconsistent contexts! Considering the undecidability results, this is more than we might expect
- False in Coq because of universe cumulativity (possibly there is an analogous statement?)

Unique typing

- We prove this by induction on the number of **alternations** between the $\Gamma \vdash e : \alpha$ and $\Gamma \vdash e \equiv e'$ judgments
- The induction hypothesis asserts that definitional inversion holds of $\vdash_n$ provability

Definition

- Let $\Gamma \vdash_0 \alpha \equiv \beta$ iff $\alpha = \beta$
- Let $\Gamma \vdash_{n+1} \alpha \equiv \beta$ iff there is a proof of $\Gamma \vdash \alpha \equiv \beta$ using only $\Gamma \vdash_n e : \alpha$ typing judgments.
- Let $\Gamma \vdash_n e : \alpha$ iff there is a proof of $\Gamma \vdash e : \alpha$ using the modified conversion rule

$$\frac{\Gamma \vdash_n e : \alpha \quad \Gamma \vdash_m \alpha \equiv \beta \quad m \leq n}{\Gamma \vdash_n e : \beta}$$

The Church Rosser theorem

Theorem (for the λ-calculus)

If $e \rightsquigarrow^* e_1$ and $e \rightsquigarrow^* e_2$, then there exists e' such that $e_1, e_2 \rightsquigarrow^* e'$.

- The Church Rosser theorem is false primarily because of proof irrelevance: there are lots of ways to prove a theorem, and they are all $\equiv$ by proof irrelevance
- Lean's reduction relation also gets stuck when η reduction interferes with the computation rule for inductives, for example:

$$\lambda h : a = a.\ \mathrm{rec}_{\mathrm{eq}\,a}\ C\ e\ a\ h \rightsquigarrow_\eta \mathrm{rec}_{\mathrm{eq}\,a}\ C\ e\ a$$
$$\lambda h : a = a.\ \mathrm{rec}_{\mathrm{eq}\,a}\ C\ e\ a\ h \rightsquigarrow_\iota \lambda h : a = a.\ e$$

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 $\ggg_\kappa$)

Future work

- More model theory of Lean (prove unprovability of $f == g \to x == y \to f\ x == g\ y$, prove that equality of types is only disprovable when the types have different cardinalities)
- Figure out how the VM evaluation model relates to the Lean reduction relation, define the type erasure map and show that VM evaluation of a well typed term gets the right answer
  - Solid theory for VM overrides?
- Prove strong normalization
- Formalize the present results in Lean

Thank you!

`https://github.com/digama0/lean-type-theory`

Future work

- More model theory of Lean (prove unprovability of $f == g \to x == y \to f\ x == g\ y$, prove that equality of types is only disprovable when the types have different cardinalities)
- Figure out how the VM evaluation model relates to the Lean reduction relation, define the type erasure map and show that VM evaluation of a well typed term gets
  - Solid theory
- Prove strong no
- Formalize the present results in Lean

Five years later...

Thank you!

`https://github.com/digama0/lean-type-theory`

~~Future~~ Past work

- My paper is now a standard reference for LeanTT
- The model theory of Lean is still not formalized, but the main results are now common knowledge (c.f. the cardinality model)
- VM reduction is still open (but see Sozeau et al.[1] in Coq)
- Strong normalization is false[2]
- **Formalize the present results in Lean**: Lean4Lean

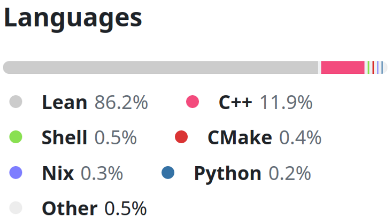
`https://github.com/digama0/lean4lean`

[1] Matthieu Sozeau, Yannick Forster, Meven Lennon-Bertrand, Jakob Botsch Nielsen, Nicolas Tabareau, et al. Correct and Complete Type Checking and Certified Erasure for Coq, in Coq (2023)

[2] Andreas Abel, Thierry Coquand. Failure of Normalization in Impredicative Type Theory with Proof-Irrelevant Propositional Equality (2020)

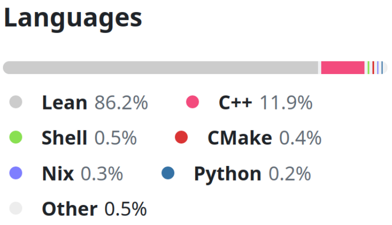
Bootstrapping Lean

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



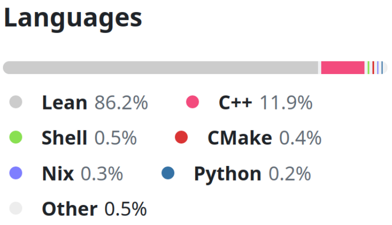
Bootstrapping Lean

- 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:

- 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.
- Write down the type theory of Lean (but formally, in Lean itself)
- Prove structural properties about the type system
- Prove the correctness of the implementation with respect to the specification

Lean4Lean

Project goals:

- ✓ 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.
- ✓ Write down the type theory of Lean (but formally, in Lean itself)
- ✓ Prove structural properties about the type system
- × 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

- Unexpected benefit: people immediately started hacking on it
  - David Renshaw: Visualizing reduction[3]
  - Rishikesh Vaishnav: Lean4Less[4]
- Lean code is much less scary than C++ for experimentation

[3] Kernel Reduction Explosion: a surprisingly inefficient computation in Lean 4 (`https://www.youtube.com/watch?v=F0t-GsiNJmU`)

[4] `https://github.com/Deducteam/lean2dk`

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

The Type Theory of Lean: Redux

$$\boxed{\Gamma \ni x : \alpha}$$

L-ZERO
$$\Gamma, x : \alpha \ni x : \alpha$$

L-SUCC
$$\frac{\Gamma \ni y : \beta}{\Gamma, x : \alpha \ni y : \beta}$$

$$\boxed{\Gamma \vdash_{E,n} e \equiv e' : \alpha}$$

$$(\Gamma \vdash e : \alpha) \triangleq (\Gamma \vdash e \equiv e : \alpha)$$

T-BVAR
$$\frac{\Gamma \ni x : \alpha}{\Gamma \vdash x : \alpha}$$

T-SYMM
$$\frac{\Gamma \vdash e \equiv e' : \alpha}{\Gamma \vdash e' \equiv e : \alpha}$$

T-TRANS
$$\frac{\Gamma \vdash e_1 \equiv e_2 : \alpha \quad \Gamma \vdash e_2 \equiv e_3 : \alpha}{\Gamma \vdash e_1 \equiv e_3 : \alpha}$$

T-SORT
$$\frac{n \vdash \ell, \ell' \text{ ok} \quad \ell \equiv \ell'}{\Gamma \vdash \mathsf{U}_\ell \equiv \mathsf{U}_{\ell'} : \mathsf{U}_{S\ell}}$$

T-CONST
$$\frac{\bar{u}.(c_{\bar{u}} : \alpha) \in E \quad \forall i,\ n \vdash \ell_i, \ell'_i \text{ ok} \ \wedge\ \ell_i \equiv \ell'_i}{\Gamma \vdash c_{\bar{\ell}} \equiv c_{\bar{\ell}'} : \alpha[\bar{u} \mapsto \bar{\ell}]}$$

T-LAM
$$\frac{\Gamma \vdash \alpha \equiv \alpha' : \mathsf{U}_\ell \quad \Gamma, x : \alpha \vdash e \equiv e' : \beta}{\Gamma \vdash (\lambda x : \alpha.\ e) \equiv (\lambda x : \alpha'.\ e') : \forall x : \alpha.\ \beta}$$

The Type Theory of Lean: Redux

T-ALL
$$\frac{\Gamma \vdash \alpha \equiv \alpha' : \mathsf{U}_{\ell_1} \quad \Gamma, x : \alpha \vdash \beta \equiv \beta' : \mathsf{U}_{\ell_2}}{\Gamma \vdash (\forall x : \alpha.\ \beta) \equiv (\forall x : \alpha'.\ \beta') : \mathsf{U}_{\mathrm{imax}(\ell_1, \ell_2)}}$$

T-APP
$$\frac{\Gamma \vdash e_1 \equiv e_1' : \forall x : \alpha.\ \beta \quad \Gamma \vdash e_2 \equiv e_2' : \alpha}{\Gamma \vdash e_1\ e_2 \equiv e_1'\ e_2' : \beta[x \mapsto e_2]}$$

T-CONV
$$\frac{\Gamma \vdash \alpha \equiv \beta : \mathsf{U}_\ell \quad \Gamma \vdash e \equiv e' : \alpha}{\Gamma \vdash e \equiv e' : \beta}$$

T-BETA
$$\frac{\Gamma, x : \alpha \vdash e : \beta \quad \Gamma \vdash e' : \alpha}{\Gamma \vdash (\lambda x : \alpha.\ e)\ e' \equiv e[x \mapsto e'] : \beta[x \mapsto e']}$$

T-ETA
$$\frac{\Gamma \vdash e : \forall y : \alpha.\ \beta}{\Gamma \vdash (\lambda x : \alpha.\ e\ x) \equiv e : \forall y : \alpha.\ \beta}$$

T-PROOF-IRREL
$$\frac{\Gamma \vdash p : \mathsf{U}_0 \quad \Gamma \vdash h : p \quad \Gamma \vdash h' : p}{\Gamma \vdash h \equiv h' : p}$$

T-EXTRA
$$\frac{\bar{u}.(e \equiv e' : \alpha) \in E \quad \forall i,\ n \vdash \ell_i \text{ ok}}{\Gamma \vdash e[\bar{u} \mapsto \bar{\ell}] \equiv e'[\bar{u} \mapsto \bar{\ell}] : \alpha[\bar{u} \mapsto \bar{\ell}]}$$

The Type Theory of Lean: Redux

- The typing judgments $\Gamma \vdash e : \alpha$ and $\Gamma \vdash e \equiv e'$ are now combined into one judgment $\Gamma \vdash e \equiv e' : \alpha$
  - $(\Gamma \vdash e : \alpha) \triangleq (\Gamma \vdash e \equiv e : \alpha)$
  - $(\Gamma \vdash e \equiv e') \triangleq \exists \alpha.\, (\Gamma \vdash e \equiv e' : \alpha)$
- (These definitions are still subject to change)

Theorems and conjectures

Theorem (Basics)

- *If* $\Gamma \vdash e \equiv e' : \alpha$*, then* e*,* e'*, and* α *are well-scoped (all free variables have indices less than* $|\Gamma|$*).*
- *If* $\Gamma, \Gamma' \vdash e \equiv e' : \alpha$*, then* $\Gamma, \Delta, \Gamma' \vdash e \equiv e' : \alpha$*.*
- *If* $\Gamma \vdash_{E,n} e \equiv e' : \alpha$ *and* $\forall i.\ n' \vdash \ell_i$*, then* $\Gamma[\bar{u} \mapsto \bar{\ell}] \vdash_{E,n'} e[\bar{u} \mapsto \bar{\ell}] \equiv e'[\bar{u} \mapsto \bar{\ell}] : \alpha[\bar{u} \mapsto \bar{\ell}]$*.*
- *If* $\Gamma, x : \beta \vdash e_1 \equiv e_2 : \alpha$ *and* $\Gamma \vdash e_0 : \beta$*, then* $\Gamma \vdash e_1[x \mapsto e_0] \equiv e_2[x \mapsto e_0] : \alpha[x \mapsto e_0]$*.*
- *If* $\Gamma, x : \alpha \vdash e_1 \equiv e_2 : \beta$ *and* $\Gamma \vdash \alpha \equiv \alpha' : \mathsf{U}_\ell$*, then* $\Gamma, x : \alpha' \vdash e_1 \equiv e_2 : \beta$*.*

Theorems and conjectures

Theorem (Inversion)

- *If* $\Gamma \vdash (\forall x : \alpha.\ \beta) : \gamma$ *then* $\Gamma \vdash \alpha$ type *and* $\Gamma, x : \alpha \vdash \beta$ type.
- *If* $\Gamma \vdash (\lambda x : \alpha.\ e) : \gamma$ *then* $\exists \beta$ *s.t.* $\Gamma \vdash \alpha$ type *and* $\Gamma, x : \alpha \vdash e : \beta$.
- *If* $\Gamma \vdash_{E,n} \mathsf{U}_\ell : \gamma$ *then* $n \vdash \ell$ ok.

$$\vdots$$

Theorems and conjectures

Theorem (Types are well-typed)

If $\Gamma \vdash e : \alpha$ *then* $\Gamma \vdash \alpha$ type.

Theorem (Substitution w.r.t both arguments)

If $\Gamma, x : \alpha \vdash f \equiv f' : \beta$ *and* $\Gamma, x : \alpha \vdash a \equiv a' : \alpha$ *then*
$\Gamma \vdash f[x \mapsto a] \equiv f'[x \mapsto a'] : \beta[x \mapsto a]$.

Theorems and conjectures

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 the proof of the above theorems, I found a gap in the proof
- I still believe the theorems are true, but the formalization process is not really clerical work at this point
- There is an alternative path to the proof of soundness, but some of the kernel optimizations depend on this theorem

Theorems and conjectures

A sneak peek at some recent results, from lemmas leading up to the main theorem:

- $\Gamma \vdash e \equiv_p e'$ is definitional equality using only η and proof irrelevance
- $\Gamma \vdash e \gg e'$ is parallel reduction
- $\Gamma \vdash e \ggg e'$ is complete parallel reduction

Strategy

✓ *$\equiv_p$ is an equivalence relation.*

✓ *If $\Gamma \vdash e : \alpha$, $e \gg e'$, and $e \ggg e^\bullet$, then there exists e° such that $\Gamma \vdash e' \gg e^\circ \equiv_p e^\bullet$.*

× *If $\Gamma \vdash e : \alpha$, and $e_1 \ll_\kappa e \gg e_2$, then $\exists e_1'\ e_2'.\ e_1 \gg e_1' \equiv_p e_2' \ll e_2$. (Church–Rosser)*

× *If $\Gamma \vdash e_1 \equiv e_2$, then $\exists e_1'\ e_2'.\ e_1 \gg e_1' \equiv_p e_2' \ll e_2$.*

Inductive types

$$\frac{\bar{u}.(e \equiv e' : \alpha) \in E \quad \forall i,\ n \vdash \ell_i \text{ ok}}{\Gamma \vdash e[\bar{u} \mapsto \bar{\ell}] \equiv e'[\bar{u} \mapsto \bar{\ell}] : \alpha[\bar{u} \mapsto \bar{\ell}]}\ \textsc{t-extra}$$

- Stuffed into the T-EXTRA rule
- Still experimenting with ways to express the rules for inductives in a way that is manageable

Inductive types

- Some successes with using a pattern language to specify the rewrite rules

$$
\begin{aligned}
p &::= \; c_{\bar{u}} \mid p\; p' \mid p\; x \\
r &::= \; e[\bar{u} \mapsto \bar{\ell}_i] \mid r\; r' \mid x_i \\
\psi &::= \; \top \mid r \equiv r' \land \psi
\end{aligned}
$$

$$
\textsc{r-pat}\;\frac{(p \rightsquigarrow r \text{ if } \psi) \in E \quad e = p[\sigma] \quad \psi[\sigma] \text{ true} \quad \forall i,\ \Gamma \vdash \sigma_i \gg \sigma'_i}{\Gamma \vdash e \gg r[\sigma]}
$$

subsumes rules like:

$$
\frac{P \text{ is SS inductive} \quad \Gamma \vdash \text{intro inv}[p,h] : \alpha \quad \Gamma \vdash C, e, p, h \gg C', e', p', h'}{\Gamma \vdash \text{rec}_P\; C\; e\; p\; h \gg e'_c\; \text{inv}[p', h']\; v'}
$$

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
- Confluence in dependent type theory is a hard problem, and (unlike Coq and Agda) in Lean we have to tackle typed reduction directly
- There are a half dozen people working on MetaCoq, but Lean doesn't have enough type theorists involved. If you identify as such, come help out!

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