Typing rules of Lean

Jeroen van Wees
Freek Wiedijk

3rd of December, 2021

Overview

Syntax

The basic grammar of Lean consist of:

$$
\begin{aligned}
l &::= u \mid 0 \mid Sl \mid \max(l, l) \mid \mathrm{imax}(l, l) \\
e &::= x \mid \mathsf{U}_l \mid ee \mid \lambda x : e.e \mid \forall x : e.e \mid \text{let } x : \alpha := e' \text{ in } e \mid c_{\overline{u}} \\
&\quad \mid \mu x : e.K \mid c_{\mu x:e.K} \mid \mathrm{rec}_{\mu x:e.K} \\
\Gamma &::= \cdot \mid \Gamma, x : e
\end{aligned}
$$

Here l represents universe levels, e expressions, and Γ context

Typing Judgements: $\Gamma \vdash e : \alpha$

The basic typing judgements of Lean are very similar to Coq:

$$\frac{}{\vdash \mathsf{U}_l : \mathsf{U}_{Sl}} \qquad \frac{\Gamma \vdash \alpha : \mathsf{U}_l \quad \Gamma \vdash e : \beta}{\Gamma, x : \alpha \vdash e : \beta} \qquad \frac{\Gamma \vdash \alpha : \mathsf{U}_l}{\Gamma, x : \alpha \vdash x : \alpha}$$

$$\ldots$$

Typing Judgements: $\Gamma \vdash e : \alpha$

The basic typing judgements of Lean are very similar to Coq:

$$\frac{}{\vdash \mathsf{U}_l : \mathsf{U}_{Sl}} \qquad \frac{\Gamma \vdash \alpha : \mathsf{U}_l \quad \Gamma \vdash e : \beta}{\Gamma, x : \alpha \vdash e : \beta} \qquad \frac{\Gamma \vdash \alpha : \mathsf{U}_l}{\Gamma, x : \alpha \vdash x : \alpha}$$

$$\ldots$$

For convenience we also have three simple judgements:

$$\frac{\Gamma \vdash \alpha : \mathsf{U}_l}{\Gamma \vdash \alpha} \qquad \frac{}{\vdash \cdot} \qquad \frac{\Gamma \vdash \alpha}{\vdash \Gamma, x : \alpha}$$

Universes

In Lean we explicitly work with something called universes.

Universes

In Lean we explicitly work with something called universes.

Each universe is identified by a level l. Using the rule $\frac{}{\vdash \mathsf{U}_l : \mathsf{U}_{Sl}}$ we see that the type of each universe is another universe of a higher level.

Universes

In Lean we explicitly work with something called universes.

Each universe is identified by a level l. Using the rule $\frac{}{\vdash \mathsf{U}_l : \mathsf{U}_{Sl}}$ we see that the type of each universe is another universe of a higher level.

The universe U_0 is equal to the set of propositions $\mathbb{P}$. Since most of mathematics is done within universe U_0 we won't worry about the details of higher universes.

Their main function for us is to prevent paradoxes in lower universes. (For example Girard's Paradox $\vdash \mathsf{U} : \mathsf{U}$).

Universes

In Lean we explicitly work with something called universes.

Each universe is identified by a level l. Using the rule $\frac{}{\vdash \mathsf{U}_l : \mathsf{U}_{Sl}}$ we see that the type of each universe is another universe of a higher level.

The universe U_0 is equal to the set of propositions $\mathbb{P}$. Since most of mathematics is done within universe U_0 we won't worry about the details of higher universes.

Their main function for us is to prevent paradoxes in lower universes. (For example Girard's Paradox $\vdash \mathsf{U} : \mathsf{U}$).

Coq also uses universes: they're called Prop, Sprop, Set and Type.

Universe levels:

The levels of universes are natural numbers, and we can compare levels with each other.

Universe levels:

The levels of universes are natural numbers, and we can compare levels with each other. There are several (trivial) rules, for example:

$$\frac{l \leq l' \quad l' \leq l}{l \equiv l'} \qquad \frac{n \geq 0}{0 \leq l + n} \qquad \frac{l_1 \leq l + n \quad l_2 \leq l + n}{\max(l_1, l_2) \leq l + n} \qquad \dots$$

Universe Levels

There is also a function called imax.
$\mathsf{imax}(m, n)$ returns 0 if $n = 0$, and $\max(m, n)$ otherwise.

$$\frac{0 \leq l + n}{\mathsf{imax}(l_1, 0) \leq l + n} \quad \frac{\max(l_1, Sl_2) \leq l + n}{\mathsf{imax}(l_1, Sl_2) \leq l + n} \quad \ldots$$

With imax we can drop back to level 0 if necessery, using this typing judgement:

$$\frac{\Gamma \vdash \alpha : \mathsf{U}_{l_1} \quad \Gamma, x : \alpha \vdash \beta : \mathsf{U}_{l_2}}{\Gamma \vdash \forall x : \alpha.\beta : \mathsf{U}_{\mathsf{imax}(l_1, l_2)}}$$

Equivalence

In Lean we have two different types of equivalence:

- Definitional Equivalence: $\alpha \equiv \beta$
- Algorithmic Equivalence: $\alpha \Leftrightarrow \beta$

Definitional Equivalence $\alpha \equiv \beta$

Definitional Equivalence is the "ideal" equivalence.

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

$$\frac{l \equiv l'}{\vdash \mathsf{U}_l \equiv \mathsf{U}_{l'}}$$

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

$$\dots$$

Definitional equivalence is unfortunatly undecidable. (This will be the topic of the next presentation.)

Algorithmic Equivalence $\alpha \Leftrightarrow \beta$

To get around the problem of decidability, we'll introduce Algorithmic Equivalence. This is what is actually checked by Lean. Since $\alpha \Leftrightarrow \beta$ implies that $\alpha \equiv \beta$, Algorithmic Equivalence is good enough.

$$\frac{}{\Gamma \vdash e \Leftrightarrow e} \qquad \frac{\Gamma \vdash e \Leftrightarrow e'}{\Gamma \vdash e' \Leftrightarrow e}$$

$$\frac{l \equiv l'}{\vdash \mathsf{U}_l \Leftrightarrow \mathsf{U}_{l'}}$$

$$\cdots$$

The rules for Algorithmic Equivalence $\alpha \Leftrightarrow \beta$ are very similar to the rules for Definintional Equivalence $\alpha \equiv \beta$
The main difference is that $\Leftrightarrow$ is not transitive, while $\equiv$ is.

let binders (ζ - reduction) and definitions (δ - reduction)

We can represent ζ-reduction by using the let binder:

$$\frac{\Gamma \vdash e' : \alpha \quad \Gamma \vdash e[e'/x] : \beta}{\Gamma \vdash \text{let } x : \alpha := e' \text{ in } e \equiv e[e'/x]}\,(\zeta) \quad \dots$$

In Lean we can also use constants and definitions (δ-reduction):

$$\frac{}{\vdash c_{\bar{l}} : \alpha[\bar{l}/\bar{u}]} \quad \frac{}{\vdash c_{\bar{l}} \equiv e[\bar{l}/\bar{u}]}\,(\delta) \quad \dots$$

Inductive Types

By Freek Wiedijk

Differences

- In Coq we don't have unique types. This is because of universe cumulativity.
- Coq uses fix and match for induction, while Lean uses rec for recursion.
- Constants/Definitions in Lean can live in variable universe levels, while in Coq their universe is less flexible
- Coq does allow nested inductive types
- Lean supports proof irrelevence, Coq asserts it as an axiom
- Lean supports quotient types
- Lean only needs three axioms (Choice, Quotient types, Propositional Extensionality), while Coq uses quite a few more