A tour in (formalised) type theory

Meven Lennon-Bertrand

Formalisation of mathematics with interactive theorem provers – 07/11/2024

✨ Formalised meta-theory of dependent types ✨

In particular:

- MetaCoq ≈ meta-theory of Coq, in Coq
- formalised logical relations

What do *you* know?

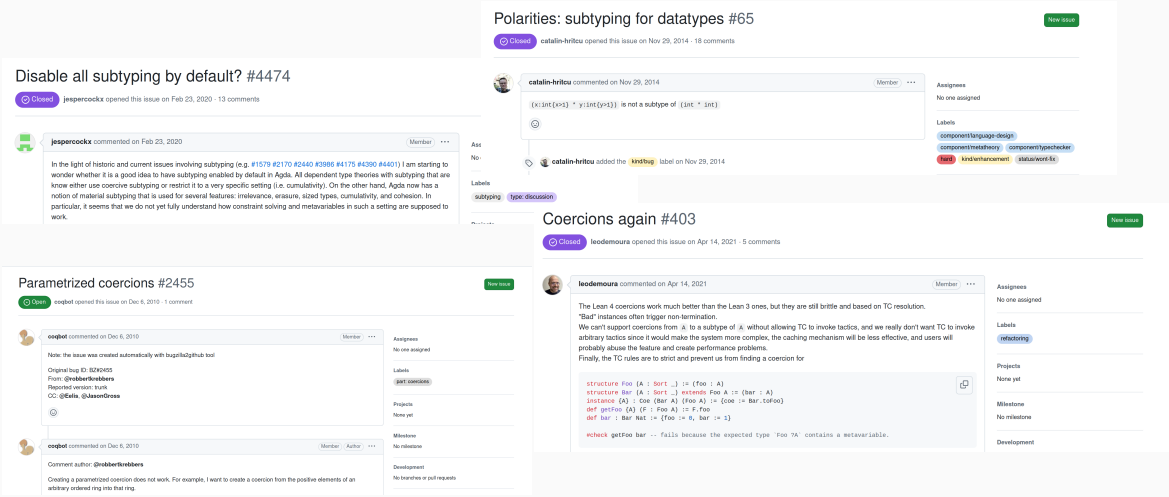
Let's start with a demo

Let's start with a demo

Summing up:

- coercions should be "transparent"!
- the current implementation does not deliver

Let's start with a demo



Let's start with a demo

Summing up:

- coercions should be "transparent"!
- the current implementation does not deliver

Solutions?

- tactics, smart typeclasses...
- understand the type theory

Dependent type theory for the impatient

Martin-Löf's logical framework

Objects:

- types/propositions $A, B \ldots$
- terms/proofs $t, u \ldots$
- contexts/hypotheses $\Gamma, \ldots$ = lists $x_1 : A_1, \ldots, x_n : A_n$

Martin-Löf's logical framework

Objects:

- types/propositions $A, B \ldots$
- terms/proofs $t, u \ldots$
- contexts/hypotheses $\Gamma, \ldots$ = lists $x_1 : A_1, \ldots, x_n : A_n$

Judgements:

$$\Gamma \vdash A \qquad \Gamma \vdash A \cong B \qquad \Gamma \vdash t : A \qquad \Gamma \vdash t \cong u : A \qquad \vdash \Gamma \qquad \vdash \Gamma \cong \Delta$$

Martin-Löf's logical framework

Objects:

- types/propositions $A, B \ldots$
- terms/proofs $t, u \ldots$
- contexts/hypotheses $\Gamma, \ldots$ = lists $x_1 : A_1, \ldots, x_n : A_n$

Judgements:

$$\Gamma \vdash A \qquad \Gamma \vdash A \cong B \qquad \Gamma \vdash t : A \qquad \Gamma \vdash t \cong u : A \qquad \vdash \Gamma \qquad \vdash \Gamma \cong \Delta$$

Rules:

$$\frac{}{x_1 : A_1, \ldots x_n : A_n \vdash x_i : A_i} \qquad \frac{\Gamma \vdash t : A \qquad \Gamma \vdash A \cong B}{\Gamma \vdash t : B}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t \cong t : A} \qquad \frac{\Gamma \vdash t \cong u : A \qquad \Gamma \vdash u \cong v : A}{\Gamma \vdash t \cong v : A} \qquad \ldots$$

Example: dependent functions

Lean: `(x : A) → B`, `fun x : A ⇒ t`, `t u`

Example: dependent functions

Lean: `(x : A) → B`, `fun x : A ⇒ t`, `t u`

$$\frac{\Gamma \vdash A \qquad \Gamma, x\colon A \vdash B}{\Gamma \vdash \Pi\, x\colon A.B} \qquad \frac{\Gamma \vdash A \qquad \Gamma, x\colon A \vdash t : B}{\Gamma \vdash \lambda\, x\colon A.t : \Pi\, x\colon A.B}$$

$$\frac{\Gamma \vdash t : \Pi\, x\colon A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\, u : B[x \leftarrow u]}$$

Example: dependent functions

Lean: `(x : A) → B`, `fun x : A ⇒ t`, `t u`

$$\frac{\Gamma \vdash A \qquad \Gamma, x\colon A \vdash B}{\Gamma \vdash \Pi\, x\colon A.B} \qquad \frac{\Gamma \vdash A \qquad \Gamma, x\colon A \vdash t : B}{\Gamma \vdash \lambda\, x\colon A.t : \Pi\, x\colon A.B}$$

$$\frac{\Gamma \vdash t : \Pi\, x\colon A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\, u : B[x \leftarrow u]}$$

$$\beta\ \frac{\Gamma, x\colon A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda\, x\colon A.t)\, u \cong t[x \leftarrow u] : B[x \leftarrow u]} \qquad \eta\ \frac{\Gamma \vdash f : \Pi\, x\colon A.B}{\Gamma \vdash f \cong \lambda x\colon A.f\ x : \Pi\, x\colon A.B}$$

Example: dependent functions

Lean: `(x : A) → B`, `fun x : A ⇒ t`, `t u`

$$\frac{\Gamma \vdash A \qquad \Gamma, x\colon A \vdash B}{\Gamma \vdash \Pi\, x\colon A.B} \qquad \frac{\Gamma \vdash A \qquad \Gamma, x\colon A \vdash t : B}{\Gamma \vdash \lambda\, x\colon A.t : \Pi\, x\colon A.B}$$

$$\frac{\Gamma \vdash t : \Pi\, x\colon A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\, u : B[x \leftarrow u]}$$

$$\beta\ \frac{\Gamma, x\colon A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda\, x\colon A.t)\, u \cong t[x \leftarrow u] : B[x \leftarrow u]} \qquad \eta\ \frac{\Gamma \vdash f : \Pi\, x\colon A.B}{\Gamma \vdash f \cong \lambda x\colon A.f\ x : \Pi\, x\colon A.B}$$

+ congruences

What else?

- (Co)inductive types: initial (final) (co)algebras for nice functors

What else?

- (Co)inductive types: initial (final) (co)algebras for nice functors
- Records: bundling data

What else?

- (Co)inductive types: initial (final) (co)algebras for nice functors
- Records: bundling data
- Universes: types for types $\dfrac{\Gamma \vdash t : \mathrm{Type}}{\Gamma \vdash \mathrm{El}(t)}$

What else?

- (Co)inductive types: initial (final) (co)algebras for nice functors
- Records: bundling data
- Universes: types for types $\dfrac{\Gamma \vdash t : \mathrm{Type}}{\Gamma \vdash \mathrm{El}(t)}$
- ...

What else?

- (Co)inductive types: initial (final) (co)algebras for nice functors
- Records: bundling data
- Universes: types for types $\dfrac{\Gamma \vdash t : \mathrm{Type}}{\Gamma \vdash \mathrm{El}(t)}$
- ...

Each time:

- types ($\Pi\, x{:}\, A.B$)
- introduction(s) ($\lambda\, x{:}\, A.B$)
- elimination(s) ($t\ u$)
- computation (β)
- uniqueness (η) (sometimes)

Aparte: how to understand this?

strings?

Aparte: how to understand this?

strings?

(first order) logic,
but with witnesses

“raw” syntax trees
+ inductive predicates

Aparte: how to understand this?

strings?

(first order) logic,
but with witnesses

"raw" syntax trees
+ inductive predicates

Abstraction

Generalised algebraic theory:

$$\mathrm{Con} : \mathrm{Set} \qquad \mathrm{Ty} : \mathrm{Con} \to \mathrm{Set} \qquad \mathrm{Tm} : \Pi(\Gamma : \mathrm{Con}).\, \mathrm{Ty}(\Gamma) \to \mathrm{Set}$$

Aparte: how to understand this?

strings?

(first order) logic,
but with witnesses

"raw" syntax trees
+ inductive predicates

Generalised algebraic theory:

$$\mathrm{Con} : \mathrm{Set} \qquad \mathrm{Ty} : \mathrm{Con} \to \mathrm{Set} \qquad \mathrm{Tm} : \Pi(\Gamma : \mathrm{Con}).\ \mathrm{Ty}(\Gamma) \to \mathrm{Set}$$

Abstraction

The good properties

- Logical consistency: there are things we cannot prove
- Decidability: we can check that a proof is valid
- Model(s): we can interpret the language in some structure we like
- Expressivity: we can express our ideas “naturally”

The good properties

- Logical consistency: there are things we cannot prove
- Decidability: we can check that a proof is valid
- Model(s): we can interpret the language in some structure we like
- Expressivity: we can express our ideas “naturally”

More technical:

- canonicity: every closed term is equal to a *value*

$$\cdot \vdash t : \mathrm{Bool} \quad \Rightarrow \quad \cdot \vdash t \cong \mathrm{true} : \mathrm{Bool} \quad \vee \quad \cdot \vdash t \cong \mathrm{false} : \mathrm{Bool}$$

- normalisation: every term is equal to a *normal form*

The good properties

- Logical consistency: there are things we cannot prove
- Decidability: we can check that a proof is valid
- Model(s): we can interpret the language in some structure we like
- Expressivity: we can express our ideas “naturally”

More technical:

- canonicity: every closed term is equal to a *value*

$$\cdot \vdash t : \mathrm{Bool} \quad \Rightarrow \quad \cdot \vdash t \cong \mathrm{true} : \mathrm{Bool} \quad \vee \quad \cdot \vdash t \cong \mathrm{false} : \mathrm{Bool}$$

- normalisation: every term is equal to a *normal form*
- subject reduction: program computation implies conversion

$$\Gamma \vdash t : T \quad \wedge \quad t \rightsquigarrow t' \quad \Rightarrow \quad \Gamma \vdash t \cong t' : T$$

Some meta-theory

How to prove consistency?

Build a model!

How to prove consistency?

Build a model!

Syntactically:

- Assume $\cdot \vdash t : \mathrm{False}$
- canonicity: there exists a value v such that $\cdot \vdash t \cong v : \mathrm{False}$
- but there are no values in False!

How to prove decidability?

$$\textsc{Conv}\ \frac{\Gamma \vdash t : A \qquad \Gamma \vdash A \cong A'}{\Gamma \vdash t : A'}$$

Conv: what is the flow of information? → bidirectional typing

How to prove decidability?

$$\textsc{Conv}\ \frac{\Gamma \vdash t : A \qquad \Gamma \vdash A \cong A'}{\Gamma \vdash t : A'} \qquad \textsc{Trans}\ \frac{\Gamma \vdash A \cong A' \qquad \Gamma \vdash A' \cong A''}{\Gamma \vdash A \cong A''}$$

Conv: what is the flow of information? → bidirectional typing

Trans: how do we guess A'?

Enter reduction

Naïve conversion checker:

$$\vdash A \cong B \quad \overset{?}{\Leftrightarrow} \quad A \rightsquigarrow \bar{A} \quad \wedge \quad B \rightsquigarrow \bar{B} \quad \wedge \quad \bar{A} \equiv_{\eta} \bar{B}$$

Enter reduction

Naïve conversion checker:

$$\vdash A \cong B \quad \overset{?}{\Leftrightarrow} \quad A \rightsquigarrow\!\!\!\!\mid\ \bar{A} \quad \wedge \quad B \rightsquigarrow\!\!\!\!\mid\ \bar{B} \quad \wedge \quad \bar{A} \equiv_{\eta} \bar{B}$$

- $\overset{?}{\Leftarrow}$ needs subject reduction

Enter reduction

Naïve conversion checker:

$$\vdash A \cong B \quad \overset{?}{\Leftrightarrow} \quad A \rightsquigarrow\!|\ \bar{A} \quad \wedge \quad B \rightsquigarrow\!|\ \bar{B} \quad \wedge \quad \bar{A} \equiv_\eta \bar{B}$$

- $\overset{?}{\Leftarrow}$ needs subject reduction
- Does $\bar{A}$ always exist? (≈ normalisation)
  - progress: if $\Gamma \vdash A$, then either A is a normal form or $\exists A'.A \rightsquigarrow A'$
  - eventually, this terminates…

Enter reduction

Naïve conversion checker:

$$\vdash A \cong B \quad \overset{?}{\Leftrightarrow} \quad A \rightsquigarrow\!\!\!\!\mid\ \bar{A} \quad \wedge \quad B \rightsquigarrow\!\!\!\!\mid\ \bar{B} \quad \wedge \quad \bar{A} \equiv_\eta \bar{B}$$

- $\overset{?}{\Leftarrow}$ needs subject reduction
- Does $\bar{A}$ always exist? (≈ normalisation)
  - progress: if $\Gamma \vdash A$, then either A is a normal form or $\exists A'.A \rightsquigarrow A'$
  - eventually, this terminates…
- uniqueness of the normal form (up to $\equiv_\eta$)?

Enter reduction

Naïve conversion checker:

$$\vdash A \cong B \quad \overset{?}{\Leftrightarrow} \quad A \rightsquigarrow\!\!\!\!\mid\ \bar{A} \quad \wedge \quad B \rightsquigarrow\!\!\!\!\mid\ \bar{B} \quad \wedge \quad \bar{A} \equiv_\eta \bar{B}$$

- $\overset{?}{\Leftarrow}$ needs subject reduction
- Does $\bar{A}$ always exist? (≈ normalisation)
  - progress: if $\Gamma \vdash A$, then either A is a normal form or $\exists A'.A \rightsquigarrow A'$
  - eventually, this terminates…
- uniqueness of the normal form (up to $\equiv_\eta$)?

Actually… 40+ years on doing this smartly!

How to prove subject reduction?

$$\Gamma \vdash (\lambda\, x{:}\, A.t)\, u : T \quad \Rightarrow \quad \begin{cases} \Gamma, x{:}\, A \vdash t : B \\ \Gamma \vdash u : A' \\ \Gamma \vdash \Pi\, x{:}\, A.B \cong \Pi\, x{:}\, A'.B' \\ \Gamma \vdash B'[x \leftarrow u] \cong T \end{cases}$$

How to prove subject reduction?

$$\Gamma \vdash (\lambda\, x\colon A.t)\, u : T \qquad \Rightarrow \qquad \begin{cases} \Gamma, x\colon A \vdash t : B \\ \Gamma \vdash u : A' \\ \Gamma \vdash \Pi\, x\colon A.B \cong \Pi\, x\colon A'.B' \\ \Gamma \vdash B'[x \leftarrow u] \cong T \end{cases}$$

$$\dfrac{\dfrac{\Gamma, x\colon A \vdash t : B' \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda\, x\colon A.t)\, u \cong t[x \leftarrow u] : B'[x \leftarrow u]}\,\beta \qquad \Gamma \vdash B'[x \leftarrow u] \cong T}{\Gamma \vdash (\lambda\, x\colon A.t)\, u \cong t[x \leftarrow u] : T}$$

How to prove subject reduction?

$$\Gamma \vdash (\lambda\, x\colon A.t)\, u : T \qquad \Rightarrow \qquad \begin{cases} \Gamma, x\colon A \vdash t : B \\ \Gamma \vdash u : A' \\ \Gamma \vdash \Pi\, x\colon A.B \cong \Pi\, x\colon A'.B' \\ \Gamma \vdash B'[x \leftarrow u] \cong T \end{cases}$$

$$\beta\ \frac{\textsc{Conv}\ \dfrac{\Gamma, x\colon A \vdash t : B \qquad \Gamma, x\colon A \vdash B \cong B'}{\Gamma, x\colon A \vdash t : B'} \qquad \textsc{Conv}\ \dfrac{\Gamma \vdash u : A' \qquad \Gamma \vdash A' \cong A}{\Gamma \vdash u : A}}{\Gamma \vdash (\lambda\, x\colon A.t)\, u \cong t[x \leftarrow u] : B'[x \leftarrow u]}$$

How to prove subject reduction?

$$\Gamma \vdash (\lambda\, x\colon A.t)\, u : T \qquad \Rightarrow \qquad \begin{cases} \Gamma, x\colon A \vdash t : B \\ \Gamma \vdash u : A' \\ \Gamma \vdash \Pi\, x\colon A.B \cong \Pi\, x\colon A'.B' \\ \Gamma \vdash B'[x \leftarrow u] \cong T \end{cases}$$

$$\beta\ \frac{\textsc{Conv}\ \dfrac{\Gamma, x\colon A \vdash t : B \qquad \Gamma, x\colon A \vdash B \cong B'}{\Gamma, x\colon A \vdash t : B'} \qquad \textsc{Conv}\ \dfrac{\Gamma \vdash u : A' \qquad \Gamma \vdash A' \cong A}{\Gamma \vdash u : A}}{\Gamma \vdash (\lambda\, x\colon A.t)\, u \cong t[x \leftarrow u] : B'[x \leftarrow u]}$$

Injectivity of Π

How to prove progress?

$$\frac{\Gamma \vdash 0 : \mathrm{Nat} \qquad \Gamma \vdash u : A \qquad \Gamma \vdash \mathrm{Nat} \cong \Pi\, x{:}\, A.B}{\Gamma \vdash 0\; u : B[x \leftarrow u]}$$

should be impossible...

How to prove progress?

$$\frac{\Gamma \vdash 0 : \mathrm{Nat} \qquad \Gamma \vdash u : A \qquad \Gamma \vdash \mathrm{Nat} \cong \Pi\, x{:}\, A.B}{\Gamma \vdash 0\; u : B[x \leftarrow u]}$$

should be impossible...

No-confusion of type constructors

The essential properties

- injectivity of type/term constructors
- no-confusion of type/term constructors
- normalisation

The essential properties

- injectivity of type/term constructors
- no-confusion of type/term constructors
- normalisation

From these we get[1] subject reduction, progress, canonicity, decidability, soundness...

[1] This is not a contractual obligation, T&C apply

The essential properties

- injectivity of type/term constructors
- no-confusion of type/term constructors
- normalisation

From these we get[1] subject reduction, progress, canonicity, decidability, soundness...

How do we show them?

[1] This is not a contractual obligation, T&C apply

Logical relations

How do we show these?

The only thing we have is typing... but normalisation cannot be proven by naïve induction!

How do we show these?

The only thing we have is typing… but normalisation cannot be proven by naïve induction!

We need a predicate $\_ \Vdash \_ : \_$ that

- “satifies” the typing rules
- implies our desired property

The predicate $\Vdash$ should be defined by looking at the type.

The predicate $\Vdash$ should be defined by looking at the type.

Core intuition for $\Gamma \Vdash t : A$

- t is good (here: normalises)
- t can only be used in a good way

The predicate $\Vdash$ should be defined by looking at the type.

Core intuition for $\Gamma \Vdash t : A$

- t is good (here: normalises)
- t can only be used in a good way

Roughly:

$$\Gamma \Vdash t : A \to B \coloneqq \begin{cases} t \text{ has a normal form } \bar{t} \\ \text{for any } u \text{ st. } \Gamma \Vdash u : A, \Gamma \Vdash \bar{t}\, u : B \end{cases}$$

But we are doing dependent types...

1. types are slippery (conversion)
2. universes: $\Gamma \Vdash t : \mathrm{Type} \coloneqq \Gamma \Vdash t ?$ $\lightning$
3. conversion rule

But we are doing dependent types...

1. types are slippery (conversion)
2. universes: $\Gamma \Vdash t : \mathrm{Type} \coloneqq \Gamma \Vdash t ?\ \lightning$
3. conversion rule

1. $\Gamma \Vdash_{\mathcal{A}} t : A$ where $\mathcal{A} : \Gamma \Vdash A$
2. careful stratification + inductive-recursive definitions
3. $\Gamma \Vdash A \cong A'$ and $\Gamma \Vdash t \cong t' : A$

But we are doing dependent types...

1. types are slippery (conversion)
2. universes: $\Gamma \Vdash t : \mathrm{Type} \coloneqq \Gamma \Vdash t ?$ ↯
3. conversion rule

1. $\Gamma \Vdash_{\mathcal{A}} t : A$ where $\mathcal{A} : \Gamma \Vdash A$
2. careful stratification + inductive-recursive definitions
3. $\Gamma \Vdash A \cong A'$ and $\Gamma \Vdash t \cong t' : A$

And much more: Kripke quantification! Partial equivalence relations! Preasheaves! Gluing!

A few proofs later

$$\frac{\Gamma \Vdash t : \Pi\, x{:}\, A.B \qquad \Gamma \Vdash u : A}{\Gamma \Vdash t\ u : B[x \leftarrow u]}$$

and many more...

A few proofs later

$$\frac{\Gamma \Vdash t : \Pi\, x{:}\, A.B \qquad \Gamma \Vdash u : A}{\Gamma \Vdash t\ u : B[x \leftarrow u]}$$

and many more...

Fundamental theorem: if $\Gamma \vdash t : A$ then $\Gamma \Vdash t : A$

$$\frac{\Gamma \Vdash t : \Pi\, x{:}\, A.B \qquad \Gamma \Vdash u : A}{\Gamma \Vdash t\ u : B[x \leftarrow u]}$$

and many more...

Fundamental theorem: if $\Gamma \vdash t : A$ then $\Gamma \Vdash t : A$

Consequences:

- $\Gamma \vdash t : A \quad \Rightarrow \quad \Gamma \Vdash t : A \quad \Rightarrow \quad t$ normalises
- $\Gamma \vdash \mathrm{Nat} \cong \Pi\, x{:}\, A.B \quad \Rightarrow \quad \Gamma \Vdash \mathrm{Nat} \cong \Pi\, x{:}\, A.B \quad \Rightarrow \quad \lightning$
- $\Gamma \vdash \Pi\, x{:}\, A.B \cong \Pi\, x{:}\, A'.B' \quad \Rightarrow \quad \Gamma \Vdash \Pi\, x{:}\, A.B \cong \Pi\, x{:}\, A'.B'$
  $\Rightarrow \quad \Gamma \Vdash A \cong A' \quad \Rightarrow \quad \Gamma \vdash A \cong A'$

$$\frac{\Gamma \Vdash t : \Pi\, x{:}\, A.B \qquad \Gamma \Vdash u : A}{\Gamma \Vdash t\ u : B[x \leftarrow u]}$$

and many more...

Fundamental theorem: if $\Gamma \vdash t : A$ then $\Gamma \Vdash t : A$

Consequences:

- $\Gamma \vdash t : A \quad \Rightarrow \quad \Gamma \Vdash t : A \quad \Rightarrow \quad t$ normalises
- $\Gamma \vdash \mathrm{Nat} \cong \Pi\, x{:}\, A.B \quad \Rightarrow \quad \Gamma \Vdash \mathrm{Nat} \cong \Pi\, x{:}\, A.B \quad \Rightarrow \quad \lightning$
- $\Gamma \vdash \Pi\, x{:}\, A.B \cong \Pi\, x{:}\, A'.B' \quad \Rightarrow \quad \Gamma \Vdash \Pi\, x{:}\, A.B \cong \Pi\, x{:}\, A'.B'$
  $\Rightarrow \quad \Gamma \Vdash A \cong A' \quad \Rightarrow \quad \Gamma \vdash A \cong A'$

Hurray!

This is not the only way

Coq in Coq:

This is not the only way

Coq in Coq:

```
Inductive term : Type :=
  | tRel (n : nat)
  | tVar (id : ident) | tEvar (ev : nat) (args : list term)
  | tLetIn (na : aname) (def : term) (def_ty : term) (body : term)
  | tSort (s : sort)
  | tProd (na : aname) (ty : term) (body : term)
  | tLambda (na : aname) (ty : term) (body : term)
  | tApp (u v : term)
  | tConst (c : kername) (u : Instance.t)
  | tInd (ind : inductive) (u : Instance.t)
  | tConstruct (ind : inductive) (idx : nat) (u : Instance.t)
  | tCase (ci : case_info) (type_info : predicate term)
      (discr : term) (branches : list (branch term))
  | tProj (proj : projection) (t : term)
  | tFix (mfix : mfixpoint term) (idx : nat)
  | tCoFix (mfix : mfixpoint term) (idx : nat)
  | tPrim (prim : prim_val term).
```

This is not the only way

Coq in Coq:

```
Inductive term : Type :=
  | tRel (n : nat)
  | tVar (id : ident) | tEvar (ev : nat) (args : list term)
  | tLetIn (na : aname) (def : term) (def_ty : term) (body : term)
  | tSort (s : sort)
  | tProd (na : aname) (ty : term) (body : term)
  | tLambda (na : aname) (ty : term) (body : term)
  | tApp (u v : term)
  | tConst (c : kername) (u : Instance.t)
  | tInd (ind : inductive) (u : Instance.t)
  | tConstruct (ind : inductive) (idx : nat) (u : Instance.t)
  | tCase (ci : case_info) (type_info : predicate term)
      (discr : term) (branches : list (branch term))
  | tProj (proj : projection) (t : term)
  | tFix (mfix : mfixpoint term) (idx : nat)
  | tCoFix (mfix : mfixpoint term) (idx : nat)
  | tPrim (prim : prim_val term).
```

MetaCoq takes another way: confluence.

+ scales very well
+ injectivity/no-confusion without normalisation
- we don't know how to treat η...

Back to subtyping

What users™ want:

$$\textsc{Sub}\ \frac{\Gamma \vdash_{\text{sub}} t : A \qquad \Gamma \vdash_{\text{sub}} A \preccurlyeq A'}{\Gamma \vdash_{\text{sub}} t : A'}$$

What users™ want:

$$\textsc{Sub}\ \frac{\Gamma \vdash_{\text{sub}} t : A \qquad \Gamma \vdash_{\text{sub}} A \preccurlyeq A'}{\Gamma \vdash_{\text{sub}} t : A'}$$

But it's a bit too wild...

Less frightening:

$$\textsc{Coe}\ \frac{\Gamma \vdash_{\mathrm{coe}} t : A \qquad \Gamma \vdash_{\mathrm{coe}} A \preccurlyeq A'}{\Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{A,A'}\, t : A'}$$

Less frightening:

$$\textsc{Coe}\ \frac{\Gamma \vdash_{\mathrm{coe}} t : A \qquad \Gamma \vdash_{\mathrm{coe}} A \preccurlyeq A'}{\Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{A,A'}\, t : A'}$$

- nicer semantics
- more reasonable for implementations

Less frightening:

$$\textsc{Coe}\ \frac{\Gamma \vdash_{\mathrm{coe}} t : A \qquad \Gamma \vdash_{\mathrm{coe}} A \preccurlyeq A'}{\Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{A,A'}\, t : A'}$$

- nicer semantics
- more reasonable for implementations

This is work for a machine!

Let's elaborate, then:

$$\textsc{Sub}\ \frac{\Gamma \vdash_{\mathrm{sub}} t : A \qquad \Gamma \vdash_{\mathrm{sub}} A \preccurlyeq A'}{\Gamma \vdash_{\mathrm{sub}} t : A'} \quad \rightsquigarrow \quad \textsc{Coe}\ \frac{\tilde{\Gamma} \vdash_{\mathrm{coe}} \tilde{t} : \tilde{A} \qquad \tilde{\Gamma} \vdash_{\mathrm{coe}} \tilde{A} \preccurlyeq \tilde{A}'}{\tilde{\Gamma} \vdash_{\mathrm{coe}} \mathrm{coe}_{\tilde{A},\tilde{A}'}\, \tilde{t} : \tilde{A}'}$$

Let's elaborate, then:

$$\textsc{Sub}\ \frac{\Gamma \vdash_{\mathrm{sub}} t : A \qquad \Gamma \vdash_{\mathrm{sub}} A \preccurlyeq A'}{\Gamma \vdash_{\mathrm{sub}} t : A'} \quad \rightsquigarrow \quad \textsc{Coe}\ \frac{\tilde{\Gamma} \vdash_{\mathrm{coe}} \tilde{t} : \tilde{A} \qquad \tilde{\Gamma} \vdash_{\mathrm{coe}} \tilde{A} \preccurlyeq \tilde{A}'}{\tilde{\Gamma} \vdash_{\mathrm{coe}} \mathrm{coe}_{\tilde{A},\tilde{A}'}\, \tilde{t} : \tilde{A}'}$$

Coherence: this should be unambiguous!

Let's elaborate, then:

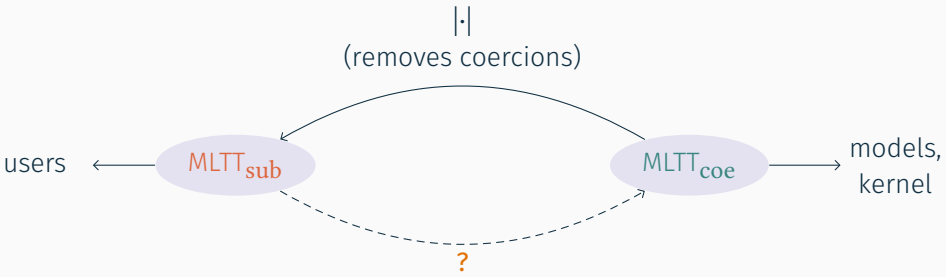
$$\textsc{Sub}\ \frac{\Gamma \vdash_{\text{sub}} t : A \qquad \Gamma \vdash_{\text{sub}} A \preccurlyeq A'}{\Gamma \vdash_{\text{sub}} t : A'} \quad \rightsquigarrow \quad \textsc{Coe}\ \frac{\tilde{\Gamma} \vdash_{\text{coe}} \tilde{t} : \tilde{A} \qquad \tilde{\Gamma} \vdash_{\text{coe}} \tilde{A} \preccurlyeq \tilde{A}'}{\tilde{\Gamma} \vdash_{\text{coe}} \operatorname{coe}_{\tilde{A},\tilde{A}'}\, \tilde{t} : \tilde{A}'}$$

Coherence: this should be unambiguous!

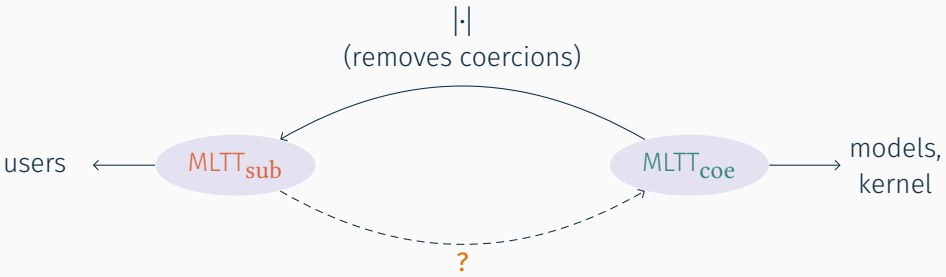
But subtyping is transitive:

$$\operatorname{coe}_{A',A''} \operatorname{coe}_{A,A'}\, t \overset{?}{\cong} \operatorname{coe}_{A,A''}\, t$$

Coherence



Coherence



? = the elaboration we want

Only well-defined if a lot of equations hold

In particular, $|t| = |u| \Rightarrow \Gamma \vdash_{\text{coe}} t \cong u : A$ (provided $\Gamma \vdash_{\text{coe}} t, u : A$)

A (vague) theorem

Laurent, L-B, Maillard (2024)

We can design a type theory $\mathrm{MLTT}_{\mathrm{coe}}$ that:

- keeps all the ✨good✨ properties
- is flexible enough to admit type-preserving elaboration from $\mathrm{MLTT}_{\mathrm{sub}}$

A (vague) theorem

Laurent, L-B, Maillard (2024)

We can design a type theory $\mathrm{MLTT}_{\mathrm{coe}}$ that:

- keeps all the ✨good✨ properties
- is flexible enough to admit type-preserving elaboration from $\mathrm{MLTT}_{\mathrm{sub}}$

The key: teach your kernel about definitional functoriality

$$\mathrm{map}\ \ g\ (\mathrm{map}\ \ f\ l) \cong \mathrm{map}\ (g \circ f)\ l$$

Thank you!