

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 \approx meta-theory of CoQ, in CoQ
- formalised logical relations



WHAT DO YOU KNOW?

LET'S START WITH A DEMO

Summing up:

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

LET'S START WITH A DEMO

Disable all subtyping by default? #4474

jespercockx opened this issue on Feb 23, 2020 - 13 comments



jespercockx commented on Feb 23, 2020

Member

In the light of historic and current issues involving subtyping (e.g. #1579 #2170 #2440 #3986 #4175 #4390 #4401) I am starting to wonder whether it is a good idea to have subtyping enabled by default in Agda. All dependent type theories with subtyping that are known either use coercive subtyping or restrict it to a very specific setting (i.e. cumulativity). On the other hand, Agda now has a notion of material subtyping that is used for several features: irrelevance, erasure, sized types, cumulativity, and cohesion. In particular, it seems that we do not yet fully understand how constraint solving and metavariables in such a setting are supposed to work.

Parametrized coercions #2455

ceobj opened this issue on Dec 6, 2010 - 1 comment



ceobj commented on Dec 6, 2010

Member

Note: the issue was created automatically with bugzilla2github tool

Original bug ID: BZ#2455
From: @robberthubbers
Reported version: trunk
CC: @Edu, @JavenDross

Assignees
No one assigned

Labels
[param-coercions](#)

Projects
None yet

Milestones
No milestone

Development
No branches or pull requests



ceobj commented on Dec 6, 2010

Member · Author

Comment author: @robberthubbers
Creating a parametrized coercion does not work. For example, I want to create a coercion from the positive elements of an arbitrary ordered ring into that ring.

Polarities: subtyping for datatypes #65

catalin-hritcu opened this issue on Nov 29, 2014 - 18 comments



catalin-hritcu commented on Nov 29, 2014

Member

`{(x:int(x>2) * y:int(y>1)) is not a subtype of ((int * int))`

catalin-hritcu added the [kind/bug](#) label on Nov 29, 2014

Assignees
No one assigned

Labels
[component/language-design](#)
[component/metadata](#) [component/typechecker](#)
[help](#) [kind/enhancement](#) [status/wont-fix](#)

Coercions again #403

leodemoura opened this issue on Apr 14, 2021 - 5 comments



leodemoura commented on Apr 14, 2021

Member

The Lean 4 coercions work much better than the Lean 3 ones, but they are still brittle and based on TC resolution. "Bad" instances often trigger non-termination.

We can't support coercions from `A` to a subtype of `A`, without allowing TC to invoke tactics, and we really don't want TC to invoke arbitrary tactics since it would make the system more complex, the caching mechanism will be less effective, and users will probably abuse the feature and create performance problems.

Finally, the TC rules are to strict and prevent us from finding a coercion for

```
structure Foo [A : Sort _] := (foo : A)
structure Bar [A : Sort _] extends Foo A := (bar : A)
instance (A) : Coe (Bar A) (Foo A) := {coe := Bar.toFoo}
def getFoo (A) (F : Foo A) := F.foo
def bar : Bar Nat := {foo := 0, bar := 1}
```

`!check getFoo bar` -- fails because the expected type 'Foo A' contains a metavariable.

Assignees
No one assigned

Labels
[refactoring](#)

Projects
None yet

Milestones
No milestone

Development

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

Objects:

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

MARTIN-LÖF'S LOGICAL FRAMEWORK

Objects:

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

Judgements:

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

MARTIN-LÖF'S LOGICAL FRAMEWORK

Objects:

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

Judgements:

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

Rules:

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

EXAMPLE: DEPENDENT FUNCTIONS

Lean: $(x : A) \rightarrow B$, $\text{fun } x : A \Rightarrow t$, $t \ u$

EXAMPLE: DEPENDENT FUNCTIONS

Lean: $(x : A) \rightarrow B$, $\text{fun } x : A \Rightarrow t$, $t u$

$$\frac{\Gamma \vdash A \quad \Gamma, x:A \vdash B}{\Gamma \vdash \Pi x:A.B}$$

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

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

EXAMPLE: DEPENDENT FUNCTIONS

Lean: $(x : A) \rightarrow B$, $\text{fun } x : A \Rightarrow t$, $t u$

$$\frac{\Gamma \vdash A \quad \Gamma, x:A \vdash B}{\Gamma \vdash \Pi x:A.B}$$

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

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

$$\beta \frac{\Gamma, x:A \vdash t:B \quad \Gamma \vdash u:A}{\Gamma \vdash (\lambda x:A.t) u \cong t[x \leftarrow u]:B[x \leftarrow u]}$$

$$\eta \frac{\Gamma \vdash f:\Pi x:A.B}{\Gamma \vdash f \cong \lambda x:A.f x:\Pi x:A.B}$$

EXAMPLE: DEPENDENT FUNCTIONS

Lean: $(x : A) \rightarrow B$, $\text{fun } x : A \Rightarrow t$, $t u$

$$\frac{\Gamma \vdash A \quad \Gamma, x:A \vdash B}{\Gamma \vdash \Pi x:A.B}$$

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

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

$$\beta \frac{\Gamma, x:A \vdash t:B \quad \Gamma \vdash u:A}{\Gamma \vdash (\lambda x:A.t) u \cong t[x \leftarrow u]:B[x \leftarrow u]}$$

$$\eta \frac{\Gamma \vdash f:\Pi x:A.B}{\Gamma \vdash f \cong \lambda x:A.f x:\Pi x:A.B}$$

+ 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

$$\frac{\Gamma \vdash t : \text{Type}}{\Gamma \vdash \text{El}(t)}$$

WHAT ELSE?

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

$$\frac{\Gamma \vdash t : \mathbf{Type}}{\Gamma \vdash \mathbf{El}(t)}$$

WHAT ELSE?

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

Each time:

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

strings?

APARTE: HOW TO UNDERSTAND THIS?

(first order) logic,
but with witnesses

strings?

“raw” syntax trees
+ inductive predicates

A
b
s
t
r
a
c
t
i
o
n



APARTE: HOW TO UNDERSTAND THIS?

strings?

(first order) logic,
but with witnesses

“raw” syntax trees
+ inductive predicates

Generalised algebraic theory:

$\text{Con} : \text{Set}$ $\text{Ty} : \text{Con} \rightarrow \text{Set}$ $\text{Tm} : \prod(\Gamma : \text{Con}). \text{Ty}(\Gamma) \rightarrow \text{Set}$

A
b
s
t
r
a
c
t
i
o
n



APARTE: HOW TO UNDERSTAND THIS?

strings?

(first order) logic,
but with witnesses

“raw” syntax trees
+ inductive predicates

Generalised algebraic theory:

$\text{Con} : \text{Set}$ $\text{Ty} : \text{Con} \rightarrow \text{Set}$ $\text{Tm} : \prod(\Gamma : \text{Con}). \text{Ty}(\Gamma) \rightarrow \text{Set}$

A
b
s
t
r
a
c
t
i
o
n



- **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”

- 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*

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

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

- 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*

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

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

HOW TO PROVE DECIDABILITY?

$$\text{CONV} \frac{\Gamma \vdash t : A \quad \Gamma \vdash A \cong A'}{\Gamma \vdash t : A'}$$

CONV: what is the flow of information? → **bidirectional typing**

HOW TO PROVE DECIDABILITY?

$$\text{CONV} \frac{\Gamma \vdash t:A \quad \Gamma \vdash A \cong A'}{\Gamma \vdash t:A'}$$

$$\text{TRANS} \frac{\Gamma \vdash A \cong A' \quad \Gamma \vdash A' \cong A''}{\Gamma \vdash A \cong A''}$$

CONV: what is the flow of information? → **bidirectional typing**

TRANS: how do we guess A' ?

Naïve conversion checker:

$$\vdash A \cong B \stackrel{?}{\Leftrightarrow} A \rightsquigarrow_{\downarrow} \bar{A} \wedge B \rightsquigarrow_{\downarrow} \bar{B} \wedge \bar{A} \equiv_{\eta} \bar{B}$$

Naïve conversion checker:

$$\vdash A \cong B \stackrel{?}{\Leftrightarrow} A \rightsquigarrow_{\downarrow} \bar{A} \wedge B \rightsquigarrow_{\downarrow} \bar{B} \wedge \bar{A} \equiv_{\eta} \bar{B}$$

- $\stackrel{?}{\Leftrightarrow}$ needs subject reduction

Naïve conversion checker:

$$\vdash A \cong B \stackrel{?}{\iff} A \rightsquigarrow \bar{A} \wedge B \rightsquigarrow \bar{B} \wedge \bar{A} \equiv_{\eta} \bar{B}$$

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

Naïve conversion checker:

$$\vdash A \cong B \stackrel{?}{\iff} A \rightsquigarrow_{\eta} \bar{A} \wedge B \rightsquigarrow_{\eta} \bar{B} \wedge \bar{A} \equiv_{\eta} \bar{B}$$

- $\stackrel{?}{\iff}$ needs subject reduction
- Does \bar{A} always exist? (\approx 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_{η})?

Naïve conversion checker:

$$\vdash A \cong B \stackrel{?}{\iff} A \rightsquigarrow_{\downarrow} \bar{A} \wedge B \rightsquigarrow_{\downarrow} \bar{B} \wedge \bar{A} \equiv_{\eta} \bar{B}$$

- $\stackrel{?}{\iff}$ needs subject reduction
- Does \bar{A} always exist? (\approx 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_{η})?

Actually... 40+ years on doing this smartly!

HOW TO PROVE SUBJECT REDUCTION?

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

HOW TO PROVE SUBJECT REDUCTION?

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

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

HOW TO PROVE SUBJECT REDUCTION?

$$\Gamma \vdash (\lambda x: A.t) u : T$$
$$\Rightarrow$$

$$\left\{ \begin{array}{l} \Gamma, x: A \vdash t : B \\ \Gamma \vdash u : A' \\ \Gamma \vdash \prod x: A.B \cong \prod x: A'.B' \\ \Gamma \vdash B'[x \leftarrow u] \cong T \end{array} \right.$$

$$\beta \frac{\text{CONV} \frac{\Gamma, x: A \vdash t : B \quad \Gamma, x: A \vdash B \cong B'}{\Gamma, x: A \vdash t : B'} \quad \text{CONV} \frac{\Gamma \vdash u : A' \quad \Gamma \vdash A' \cong A}{\Gamma \vdash u : A}}{\Gamma \vdash (\lambda x: A.t) u \cong t[x \leftarrow u] : B'[x \leftarrow u]}$$

HOW TO PROVE SUBJECT REDUCTION?

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

$$\beta \frac{\text{CONV} \frac{\Gamma, x: A \vdash t : B \quad \Gamma, x: A \vdash B \cong B'}{\Gamma, x: A \vdash t : B'} \quad \text{CONV} \frac{\Gamma \vdash u : A' \quad \Gamma \vdash A' \cong A}{\Gamma \vdash u : A}}{\Gamma \vdash (\lambda x: A.t) u \cong t[x \leftarrow u] : B'[x \leftarrow u]}$$

Injectivity of Π

HOW TO PROVE PROGRESS?

$$\frac{\Gamma \vdash 0 : \text{Nat} \quad \Gamma \vdash u : A \quad \Gamma \vdash \text{Nat} \cong \prod x : A. B}{\Gamma \vdash 0 u : B[x \leftarrow u]}$$

should be impossible...

$$\frac{\Gamma \vdash 0 : \text{Nat} \quad \Gamma \vdash u : A \quad \Gamma \vdash \text{Nat} \cong \prod x : A. B}{\Gamma \vdash 0 u : B[x \leftarrow u]}$$

should be impossible...

No-confusion of type constructors

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

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

From these we get¹ subject reduction, progress, canonicity, decidability, soundness...

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

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

From these we get¹ subject reduction, progress, canonicity, decidability, soundness...

How do we show them?

¹This 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

- “satisfies” 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:

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

BUT WE ARE DOING DEPENDENT TYPES...

1. types are slippery (conversion)
2. universes: $\Gamma \Vdash t : \mathbf{Type} := \Gamma \Vdash t? \downarrow$
3. conversion rule

BUT WE ARE DOING DEPENDENT TYPES...

1. types are slippery (conversion)
2. universes: $\Gamma \Vdash t : \mathbf{Type} := \Gamma \Vdash t? \not\Leftarrow$
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: $\Gamma \Vdash t : \mathbf{Type} := \Gamma \Vdash t? \not\Leftarrow$
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!

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

and many more...

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

and many more...

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

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

and many more...

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

Consequences:

- $\Gamma \vdash t : A \Rightarrow \Gamma \Vdash t : A \Rightarrow t$ normalises
- $\Gamma \vdash \text{Nat} \cong \Pi x : A. B \Rightarrow \Gamma \Vdash \text{Nat} \cong \Pi x : A. B \Rightarrow \zeta$
- $\Gamma \vdash \Pi x : A. B \cong \Pi x : A'. B' \Rightarrow \Gamma \Vdash \Pi x : A. B \cong \Pi x : A'. B'$
 $\Rightarrow \Gamma \Vdash A \cong A' \Rightarrow \Gamma \vdash A \cong A'$

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

and many more...

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

Consequences:

- $\Gamma \vdash t : A \Rightarrow \Gamma \Vdash t : A \Rightarrow t$ normalises
- $\Gamma \vdash \text{Nat} \cong \Pi x : A. B \Rightarrow \Gamma \Vdash \text{Nat} \cong \Pi x : A. B \Rightarrow \zeta$
- $\Gamma \vdash \Pi x : A. B \cong \Pi x : A'. B' \Rightarrow \Gamma \Vdash \Pi x : A. B \cong \Pi x : A'. B'$
 $\Rightarrow \Gamma \Vdash A \cong A' \Rightarrow \Gamma \vdash A \cong A'$

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:

$$\text{SUB} \frac{\Gamma \vdash_{\text{sub}} t : A \quad \Gamma \vdash_{\text{sub}} A \preccurlyeq A'}{\Gamma \vdash_{\text{sub}} t : A'}$$

What users™ want:

$$\text{SUB} \frac{\Gamma \vdash_{\text{sub}} t : A \quad \Gamma \vdash_{\text{sub}} A \preccurlyeq A'}{\Gamma \vdash_{\text{sub}} t : A'}$$

But it's a bit too wild...

Less frightening:

$$\text{COE} \frac{\Gamma \vdash_{\text{coe}} t : A \quad \Gamma \vdash_{\text{coe}} A \preccurlyeq A'}{\Gamma \vdash_{\text{coe}} \text{coe}_{A,A'} t : A'}$$

Less frightening:

$$\text{COE} \frac{\Gamma \vdash_{\text{coe}} t : A \quad \Gamma \vdash_{\text{coe}} A \preccurlyeq A'}{\Gamma \vdash_{\text{coe}} \text{coe}_{A,A'} t : A'}$$

- nicer semantics
- more reasonable for implementations

Less frightening:

$$\text{COE} \frac{\Gamma \vdash_{\text{coe}} t : A \quad \Gamma \vdash_{\text{coe}} A \preccurlyeq A'}{\Gamma \vdash_{\text{coe}} \text{coe}_{A,A'} t : A'}$$

- nicer semantics
- more reasonable for implementations

This is work for a machine!

Let's elaborate, then:

$$\text{SUB} \frac{\Gamma \vdash_{\text{sub}} t : A \quad \Gamma \vdash_{\text{sub}} A \preccurlyeq A'}{\Gamma \vdash_{\text{sub}} t : A'} \rightsquigarrow \text{COE} \frac{\tilde{\Gamma} \vdash_{\text{coe}} \tilde{t} : \tilde{A} \quad \tilde{\Gamma} \vdash_{\text{coe}} \tilde{A} \preccurlyeq \tilde{A}'}{\tilde{\Gamma} \vdash_{\text{coe}} \text{coe}_{\tilde{A}, \tilde{A}'} \tilde{t} : \tilde{A}'}$$

Let's elaborate, then:

$$\text{SUB} \frac{\Gamma \vdash_{\text{sub}} t : A \quad \Gamma \vdash_{\text{sub}} A \preccurlyeq A'}{\Gamma \vdash_{\text{sub}} t : A'} \rightsquigarrow \text{COE} \frac{\tilde{\Gamma} \vdash_{\text{coe}} \tilde{t} : \tilde{A} \quad \tilde{\Gamma} \vdash_{\text{coe}} \tilde{A} \preccurlyeq \tilde{A}'}{\tilde{\Gamma} \vdash_{\text{coe}} \text{coe}_{\tilde{A}, \tilde{A}'} \tilde{t} : \tilde{A}'}$$

Coherence: this should be unambiguous!

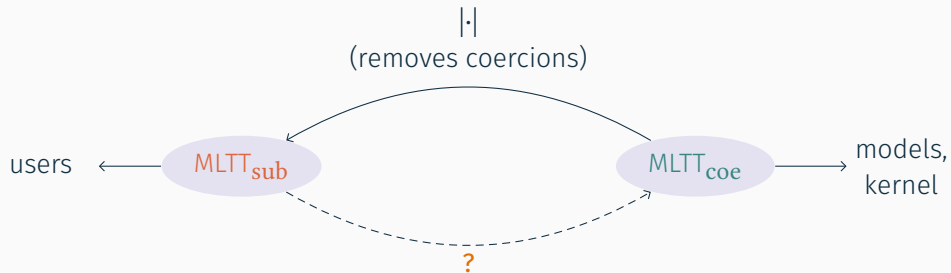
Let's elaborate, then:

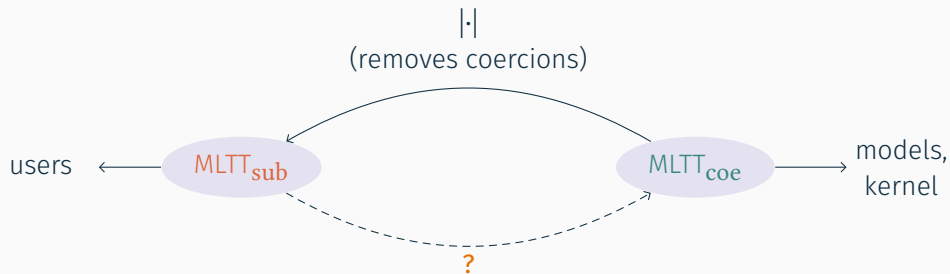
$$\text{SUB} \frac{\Gamma \vdash_{\text{sub}} t : A \quad \Gamma \vdash_{\text{sub}} A \preccurlyeq A'}{\Gamma \vdash_{\text{sub}} t : A'} \rightsquigarrow \text{COE} \frac{\tilde{\Gamma} \vdash_{\text{coe}} \tilde{t} : \tilde{A} \quad \tilde{\Gamma} \vdash_{\text{coe}} \tilde{A} \preccurlyeq \tilde{A}'}{\tilde{\Gamma} \vdash_{\text{coe}} \text{coe}_{\tilde{A}, \tilde{A}'} \tilde{t} : \tilde{A}'}$$

Coherence: this should be unambiguous!

But subtyping is transitive:

$$\text{coe}_{A', A''} \text{coe}_{A, A'} t \stackrel{?}{\cong} \text{coe}_{A, A''} t$$





$?$ = the elaboration we want

Only well-defined if a lot of equations hold

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

Laurent, L-B, Maillard (2024)

We can design a type theory $MLTT_{coe}$ that:

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

A (VAGUE) THEOREM

Laurent, L-B, Maillard (2024)

We can design a type theory MLTT_{coe} that:

- keeps all the ✨ good ✨ properties
- is flexible enough to admit type-preserving elaboration from MLTT_{sub}

The key: teach your kernel about **definitional functoriality**

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

THANK YOU!