The Coherence Problem in HoTT

Thorsten Altenkirch
FP Away Day 2014
Before the revolution...

• In Intensional Type Theory the equality type

\[
\text{data } _\equiv_ \ {\{A : \text{Set}\} (x : A) : A \to \text{Set} \ where}
\text{refl} : x \equiv x
\]

• reflects definitional equality

• is proof-irrelevant

• is not extensional:

  • does not validate functional extensionality

  • does not validate univalent
...after the revolution

- In HoTT the equality type:
  - does not reflect propositional equality
  - is proof relevant
  - is extensional
    - validates functional extensionality
    - validates univalence
Are we happy now?

Univalent equality is what you want to do Mathematics!

But sometimes I would like to have a strict equality …

Vladimir Voevodsky
Voevodsky's exercise

Define
Semisimplicial
Types in HoTT!
Semisimplicial Types

SSType = \(\Sigma\)

\((\mathcal{X}_0 : U)\)

\((\mathcal{X}_1 : \mathcal{X}_0 \to \mathcal{X}_0 \to U)\)

\((\mathcal{X}_2 : \{\mathcal{X}_0 \mathcal{X}_1 \mathcal{X}_2 : \mathcal{X}_0\})\)

\(\to \mathcal{X}_1 \mathcal{X}_0 \mathcal{X}_1 \to \mathcal{X}_1 \mathcal{X}_1 \mathcal{X}_2 \to \mathcal{X}_1 \mathcal{X}_0 \mathcal{X}_2 \to U\)

\((\mathcal{X}_3 : \{\mathcal{X}_0 \mathcal{X}_1 \mathcal{X}_2 \mathcal{X}_3 : \mathcal{X}_0\}\{\mathcal{X}_0_1 : \mathcal{X}_1 \mathcal{X}_0 \mathcal{X}_1\}\{\mathcal{X}_1_2 : \mathcal{X}_1 \mathcal{X}_1 \mathcal{X}_2\}\{\mathcal{X}_0_2 : \mathcal{X}_1 \mathcal{X}_0 \mathcal{X}_2\}\{\mathcal{X}_0_3 : \mathcal{X}_1 \mathcal{X}_0 \mathcal{X}_3\}\{\mathcal{X}_1_3 : \mathcal{X}_1 \mathcal{X}_1 \mathcal{X}_3\}\{\mathcal{X}_2_3 : \mathcal{X}_1 \mathcal{X}_2 \mathcal{X}_3\})\)

\(\to \mathcal{X}_2 \mathcal{X}_0_1 \mathcal{X}_1_2 \mathcal{X}_0_2\)

\(\to \mathcal{X}_2 \mathcal{X}_0_1 \mathcal{X}_1_3 \mathcal{X}_0_3\)

\(\to \mathcal{X}_2 \mathcal{X}_0_2 \mathcal{X}_2_3 \mathcal{X}_0_3\)

\(\to \mathcal{X}_2 \mathcal{X}_1_2 \mathcal{X}_2_3 \mathcal{X}_1_3\)

\(\to U\)

\[\ldots\]
SSType in old Type Theory

record Δ (m n : ℕ) : U where
  field
    f : Fin (suc m) → Fin (suc n)
    isMonotone : monotone f
    isInjective : injective f

record SSet : U₁ where
  field
    X : ℕ → U
    Xm : ∀ {m}{n} → Δ m n → X n → X m
    Xid : ∀{m}{x : X m} → Xm idΔ x ≡ x
    Xo : ∀ {l}{m}{n}{f : Δ m n}{g : Δ l m}{x : X n} → Xm (f oΔ g) x ≡ Xm g (Xm f x)
SSType in HoTT?

- This does not work in HoTT.
- Equality is proof-relevant!
- Hence to be equivalent to the context we need to add coherence laws.
Coherence Laws?

E.g. There are two ways to prove $\text{Xm} \left( f \circ \Delta \ id \right) x \equiv \text{Xm} f x$ :

1. Using $f \circ \Delta \ id \equiv f$ and that $\text{Xm}$ preserves equality.
2. Using $\text{Xo}$ : $\text{Xm} \left( f \circ \Delta \ id \right) x \equiv \text{Xm} \ id \ \Delta \ \left( \text{Xm} f x \right)$
   and $\text{Xid}$ : $\text{Xm} \ id \ \Delta \ \left( \text{Xm} f x \right) \equiv \text{Xm} f x$
   and transitivity.

And we need them to be equal!
Coherence laws …

• There are infinitely many such laws at higher dimensions.

• Defining the type of coherence laws doesn’t seem easier than defining SSType itself!
Genius needed!

• But maybe there is another way to define SSet avoiding the coherence problem!
• E.g. can’t we define the approximations using recursion?
• Many have tried …

• But nobody has succeeded!
Strict equality?

• But do we really need to solve a coherence problem to define SSType?

• We want the equalities in the presheaf definition to be strict!

• So that the approximations are strictly isomorphic to the corresponding contexts.
Strict Equality ??

- We would like to have access to a strict equality that \textit{reflects} definitional equality.
- Let’s write $=$ for the extensional equality from HoTT and $\equiv$ for the strict equality.
- But we cannot have two different equalities because we can show $a = b \rightarrow a \equiv b$.
I want my own Type Theory where I can have both equalities. I call it HTS (for Homotopy Type System).

I am going to implement it!

Dan Grayson
HTS

• Features extensional equality (=) and strict equality (≡)

• Strict equality uses equality reflection (as in NuPRL).

• Hence type-checking is undecidable.

• Distinguishes between pretypes (like a ≡ b) and types (like a = b).

• Extensional equality can only eliminate over types.

• Hence a = b does not entail a ≡ b.
An alternative to HTS

- Since HTS is based on Extensional Type Theory it cannot be easily simulated in Agda or Coq.

- I propose an alternative which I have implemented in Agda

- It also differs from HTS in that we can define dependent types from fibrations.

- I am not yet sure whether I can already define semisimplicial types.
A universe ...

data U : Set
El : U → Set

data U where
  UU : U
  π : (A : U) (B : El A → U) → U
  σ : (A : U) (B : El A → U) → U
  Nat : U
  _~_ : {A : U} → El A → El A → U

El UU = U
El (π A B) = (x : El A) → El (B x)
El (σ A B) = Σ (El A) (λ x → El (B x))
El Nat = N
El (a ~ a') = ...
... with extensional equality

\[
\begin{align*}
\text{refl}~ & : \{A : U\}\{x : \text{El } A\} \rightarrow \text{El } (x \sim x) \\
\text{J}~ & : \{A : U\}\{a : \text{El } A\} \\
& \quad (P : \{a' : \text{El } A\} \rightarrow \text{El } (a \sim a') \rightarrow U) \\
& \quad \rightarrow \text{El } (P \ \text{refl}\sim) \\
& \quad \rightarrow \{a' : \text{El } A\}(p : \text{El } (a \sim a')) \rightarrow \text{El } (P \ p) \\
\text{ext} & : \{A : U\}\{B : \text{El } A \rightarrow U\}\{f f' : \text{El } (\pi A B)\} \\
& \rightarrow (\{x : \text{El } A\} \rightarrow \text{El } ((f \ x) \sim (f' \ x))) \\
& \rightarrow \text{El } (f \sim f') \\
\text{ua} & : \{A B : \text{El } \text{UU}\}\{p : \text{El } (A \sim B)\} \rightarrow \text{isEquiv } (\text{coe } p)
\end{align*}
\]
• Here Agda’s \( \equiv \) plays the role of strict equality.

• \( \sim \) represents extensional equality.

• Agda types correspond to pretypes.

• While elements of U correspond to proper (extensional) types.

• We cannot prove \( a \sim b \) implies \( a \equiv b \) because \( J\sim \) only eliminates over types.
What is a fibration?

isFib : \{A B : U\}(f : El A \rightarrow El B) \rightarrow Set

isFib {A} {B} f = (a : El A)(b : El B)(p : El (f a ~ b))
    \rightarrow \Sigma[ a' \in El A ]
    \Sigma[ q \in El (a ~ a') ]
    \_\_\_ (A = \Sigma[ b' \in El B ]
    El (f a ~ b'))
    (b , p)
    ((f a') , cong~ f q)
Projections are fibrations

\[
\text{fst} : \{A : U\}\{B : \text{El } A \to U\} \to \text{El } (\sigma A B) \to \text{El } A
\]
\[
\text{fst} (a, b) = a
\]

\[
\text{isFibFst} : \{A : U\}\{B : \text{El } A \to U\} \to \text{isFib } (\text{fst } \{A\} \{B\})
\]

We need an extra assumption:

\[
p \equiv \text{cong\sim } \text{fst } \text{cong\sim } [p, q]
\]
A new type former

data U where

...  
Fam : {A B : U}(f : El A → El B) → isFib f → El B → U

El (Fam {A} f _ b) = Σ[ a ∈ El A ] f a ≡ b
Conclusions

• We can show that types are closed under strict pullbacks.

• We can also define strict versions of certain presheaf categories.

• Can we define SSType?