Towards Higher Observational Type Theory

From parametricity to identity types

Thorsten Altenkirch

Functional Programming Laboratory
School of Computer Science
University of Nottingham

jww Ambrus Kaposi, Michael Shulman, Elif Üsküplü and Szumi Xie

June 9, 2025

What is equality?

```
data _≡_ {A : Set} : A → A → Set where
  refl : {a : A} → a ≡ a
```

- The inductive definition doesn't **define** equality.
- It reflects equality.
- Intensional Type Theory: equality is definitional identity.

HOTT Vision

- A type has elements.
- Every type comes with an equality type.
- Every function preserves equality (apply path)
- We can coerce between equal types (transport)
- Equality is *observational*.

Towards HOTT

1. A theory of parametricity (POTT) [POPL24, TYPES24]
2. Narya (implemented by Mike and Elif).
3. Define fibrant types using a higher coinductive type in Narya.
4. Show that type formers are fibrant.

Narya : natural numbers

```
def ℕ : Type ≔ data [ zero. | suc. (_ : ℕ) ]

def add (m n : ℕ) : ℕ ≔
    match m [ zero. ↦ n
            | suc. m ↦ suc. (add m n) ]
```

Bridge types

```
zero. :  ℕ
zero. : Br ℕ zero. zero.

suc. :  ℕ →  ℕ
suc. : {m n : ℕ} → Br ℕ m n → Br ℕ (suc. m) (suc. n)

def add0 (m : ℕ) : Br ℕ (add m zero.) m ≔
match m [
| zero. ↦ zero.
| suc. m ↦ suc. (add0 m)]
```

Streams

```
def Stream (A : Type) : Type ≔
   codata [ x .head : A
 | x .tail : Stream A ]

def from (n : ℕ) : Stream ℕ ≔
  [ .head ↦ n | .tail ↦ from (suc. n) ]

def map_suc (ns : Stream ℕ) : Stream ℕ ≔ [
| .head ↦ suc. (ns .head)
| .tail ↦ map_suc (ns .tail)]

def thm (n : ℕ) :
  Br (Stream ℕ) (from (suc. n)) (map_suc (from n))≔ [
| .head ↦ suc. (rel n)
| .tail ↦ thm (suc n)]
```

Bridges in the universe

```
A₀ A₁ : U
A₂ : Br U A₀ A₁

a₀ : A₀
a₁ : A₁
A₂ a₀ a₁ : U

def Gel (A B : Type) (R : A → B → Type) :
      Br Type A B ≔ sig a b ↦ (
                      ungel : R a b )

Gel (a₀ a₁ ↦ A₂ a₀ a₁) = A₂ ?
```

not supported by the cubical semantics.

Fibrant types (as a higher coinductive type)

```
def isFib (A : Type) : Type := codata [
   | x .trr.p   : A.0 → A.1
   | x .trl.p   : A.1 → A.0
   | x .liftr.p : (a0 : A.0) → A.2 a0 (x.2 .trr a0)
   | x .liftl.p : (a1 : A.1) → A.2 (x.2 .trl a1) a1
   | x .id.p    : (a0 : A.0) (a1 : A.1)
                  → isFib (A.2 a0 a1) ]

def Fib : Type ≔
    sig ( t : Type, f : isFib t )
```

Bridges in Fib $\cong$ equivalences

```
(A₂,f₂) : Br Fib (A₀,f₀) (A₂,f₁)

A₂ : Br U A₀ A₁
f₂ .trr : A₀ → A₁
f₂ .liftr : (a₀ : A₀) → A₂ a₀ (f₂ .trr a₀)
```

Fibrant type constructors

Example `A × B`

```
def fprod (A B : Type)
          (fA : isFib A) (fB : isFib B)
  : isFib (A × B)
```

⋮

```
| .trr.p ↦ u0 ↦
   (fA.2 .trr (u0 .fst), fB.2 .trr (u0 .snd))
| .liftr.p ↦ u0 ↦
   (fA.2 .liftr (u0 .fst), fB.2 .liftr (u0 .snd))
```

To prove `.id.p` we use that (wrt extensional equality):

```
A2 a0 a1 × B2 b0 b1 ≅ Br (A2 × B2) (a0, b0) (a1, b1)
```

And that `isFib` is preserved by ≅.

Fibrant types

- $0, 1, 2$
- Π-types
- Σ-types
- Indexed W-types
- Indexed M-types

Is Fib fibrant?

- We can define a fibrant equivalent of bridges in the universe: equivalence.
- We can prove logical equivalence (encode, decode).
- But we cannot show that is a retract!
- We need `Gel (a₀ a₁ ↦ A₂ a₀ a₁) = A₂`
- Revisit the semantics for the parametricity calculus?!

Future plans

- Semantics for higher coinductive types
- Implement a native version of HOTT.
- Add higher inductive types
- Semicubical types (fibrant ?)