Coinductive Universes and Higher Observational Type Theory

Michael Shulman

University of San Diego

jww Thorsten Altenkirch, Ambrus Kaposi, Szumi Xie

April 24, 2026

OCIE and MPP Seminar

Chapman University

Outline

Review of MLTT

Martin-Löf type theory is an alternative foundation for mathematics.

- Based on types, which are kind of like sets.
- Types don't overlap: every element has exactly one type.
- The elements of a type are introduced along with it:
  $A \times B$ contains pairs, $A \to B$ contains functions, etc.
- Types also represent propositions, inhabited when true.
- A universe type $\mathcal{U}$ contains types as its elements.
- $B : A \to \mathcal{U}$ is a type family or predicate.
- Used in many computer proof assistants (Rocq, Agda, Lean).

Justification of MLTT

When we define each type former, we specify both

- how to construct or introduce its elements, and
- how to use or eliminate its elements.

These two sets of rules are in harmony:

- The elimination rules specify exactly how they are to behave when applied to elements constructed by the introduction rules.
- The introduction rules specify exactly how an element is to behave when the elimination rules are applied to it.

E.g. we are "justified" in defining functions on $\mathbb{N}$ by recursion (elimination), since such a definition specifies how to compute it on numerals constructed from zero and successor (introduction).

Harmony makes MLTT a programming language we can execute.

Review of HoTT

Homotopy type theory is a modern perspective on MLTT.

- In MLTT, the proposition "$a = b$" is a type $\mathsf{Id}_A(a, b)$.
- The standard rules allow > 1 element in $\mathsf{Id}_A(a, b)$.
  Groupoid and homotopical models realize this possibility.
- The rules of Id make every type behave like a groupoid/space.
- The univalence principle implies that $\mathsf{Id}_{\mathcal{U}}(A, B) \cong (A \cong B)$:
  two types are equal just when they are isomorphic.
- Structural mathematics: ensures global isomorphism-invariance.
- Permits synthetic homotopy theory and higher category theory.

The problem of HoTT

How do we define $\mathsf{Id}_A(a, b)$?

- In original MLTT, Id_A is the smallest reflexive relation on A.
- Entails all the higher groupoid structure automatically.
- Incompletely specified: $\mathsf{Id}_{A \to B}$ and $\mathsf{Id}_{\mathcal{U}}$ undetermined.
- The original "Book" HoTT adds univalence as an axiom. This is no longer justified by harmony, and in particular it blocks execution of type theory as a programming language.

"Cubical type theory" makes univalence compute:

- Defines Id as paths mapping out of an "interval".
- Builds in the higher groupoid structure explicitly.
- Relies on higher mathematics; hard to justify philosophically.
- Difficult to implement and use in practice.

Towards higher observational type theory

In Observational Type Theory, Id_A is defined based on A.

- $\mathrm{Id}_{A\times B}((a_1, b_1), (a_2, b_2)) = \mathrm{Id}_A(a_1, a_2) \times \mathrm{Id}_B(b_1, b_2)$: pairs are equal if their components are equal.
- Functions are equal if they map equal inputs to equal outputs.
- ...

Higher Observational Type Theory combines this with univalence:

- $\mathrm{Id}_{\mathcal{U}}(A, B) = (A \cong B)$ by definition.

How do we justify this, philosophically and mathematically?

Outline

Induction

Some types are defined inductively or positively:
First we specify the introduction rules, and then the elimination rules are determined by how they act on the introductions.

- The disjoint union $A \sqcup B$ is introduced by including A and B.
  We define functions out of it by cases.
- $\mathbb{N}$ is introduced by $0 : \mathbb{N}$ and $s : \mathbb{N} \to \mathbb{N}$.
  We define functions on $\mathbb{N}$ by recursion.
- $\mathsf{List}(A)$ by $\mathsf{nil} : \mathsf{List}(A)$ and $\mathsf{cons} : A \times \mathsf{List}(A) \to \mathsf{List}(A)$.
  We define functions on it by list recursion.

Coinduction

Other types are defined coinductively or negatively:
First we specify the elimination rules, and then the introduction rules are determined by how the elimination rules act on them.

- $A \times B$ is eliminated by projections to A and B.
  We introduce a pair by specifying what its projections are.
- Stream(A) is eliminated by

$$\begin{aligned} \text{head} &: \text{Stream}(A) \to A \\ \text{tail} &: \text{Stream}(A) \to \text{Stream}(A) \end{aligned}$$

  We introduce an infinite stream by giving a process that generates an element at each step. For example:

$$\begin{aligned} \text{head}(\text{ones}) &= 1 \\ \text{tail}(\text{ones}) &= \text{ones}. \end{aligned}$$

  yields ones $= (1, 1, 1, 1, \dots)$.

The mysteries of the universe

What sort of a type is $\mathcal{U}$?

The type formers are probably introduction rules:

- If $A : \mathcal{U}$ and $B : \mathcal{U}$, then $A \times B : \mathcal{U}$, so $(\times) : \mathcal{U} \times \mathcal{U} \to \mathcal{U}$.
- If $A : \mathcal{U}$ and $B : \mathcal{U}$, then $A \to B : \mathcal{U}$, so $(\to) : \mathcal{U} \times \mathcal{U} \to \mathcal{U}$.
- $\mathbb{N} : \mathcal{U}$.

One candidate for an elimination rule is the Tarski eliminator:

- For any $A : \mathcal{U}$, there is a collection $\mathrm{El}(A)$ of "elements of A".

This makes more sense if we think of elements of $\mathcal{U}$ as "codes for types", with El a "decoding function".

But these don't "harmonize" inductively or coinductively.

Inductive universes

If $\mathcal{U}$ were inductively defined by the type formers as constructors, then in addition to El we could define other operations by "universe recursion" or "type-case". For example, identity types:

$$\begin{aligned}
\mathsf{Id}_{A\times B}((a_1,b_1),(a_2,b_2)) &= \mathsf{Id}_A(a_1,a_2)\times \mathsf{Id}_B(b_1,b_2)\\
\mathsf{Id}_{A\to B}(f,g) &= (x:A)\to \mathsf{Id}_B(f(x),g(x))\\
&= \prod_{x:A} \mathsf{Id}_B(f(x),g(x))\\
&= \forall (x:A), \mathsf{Id}_B(f(x),g(x))\\
&\vdots
\end{aligned}$$

This perspective leads to (lower) Observational Type Theory.

NB: the universe in OTT is not inductively defined inside the type theory; we simply think of it that way, meta-theoretically.

Inductive universes are not univalent

Lower Observational Type Theory is not univalent. Indeed, identity types of inductive types are determined already by MLTT.

$$\mathsf{Id}_{\mathbb{N}}(0,0) = \mathbf{1} \qquad \mathsf{Id}_{\mathbb{N}}(0,s(n)) = \mathbf{0}$$
$$\mathsf{Id}_{\mathbb{N}}(s(m),s(n)) = \mathsf{Id}_{\mathbb{N}}(m,n) \qquad \mathsf{Id}_{\mathbb{N}}(s(n),0) = \mathbf{0}.$$

Therefore, in OTT:

$$\begin{aligned}
\mathsf{Id}_{\mathcal{U}}(A \times B, C \times D) &= \mathsf{Id}_{\mathcal{U}}(A,C) \times \mathsf{Id}_{\mathcal{U}}(B,D) \\
\mathsf{Id}_{\mathcal{U}}(A \to B, C \to D) &= \mathsf{Id}_{\mathcal{U}}(A,C) \times \mathsf{Id}_{\mathcal{U}}(B,D) \\
\mathsf{Id}_{\mathcal{U}}(A \times B, C \to D) &= \mathbf{0} \\
&\vdots
\end{aligned}$$

Whereas, e.g., $(1 \times 1) \cong (1 \to 1)$, so univalence would equate them.

Inductive universes are closed

An inductive universe is also a closed universe.

- When we define such a universe, we have to specify a particular collection of type formers ($\times$, $\to$, ...) as its constructors.
- The universe is closed under those type formers and no others.
- If we discover or invent a new type former, we have to redefine the universe to include it.

In addition:

- The recursive definition of identity types is a separate step from the definition of the type formers that constitute the universe. Identity types are not intrinsic to each type.

Cf. "the expression problem"

Coinductive universes

Instead we can consider $\mathcal{U}$ to be coinductively defined by El and Id.

1. For any $A : \mathcal{U}$, there is a collection of "elements of A".
2. For any $A : \mathcal{U}$, there is a family of types $\mathsf{Id}_A(a, b) : \mathcal{U}$.

Note "Id" is "corecursive" like "tail", producing more elements of $\mathcal{U}$. Just as "tail" produces an infinite stream of elements, "Id" produces an infinite tower of identity types.

Now the introduction rule is determined: we define $A : \mathcal{U}$ by giving

1. Its collection of elements, and
2. For any two such elements a, b, a type $\mathsf{Id}_A(a, b) : \mathcal{U}$.

Type formers, coinductively

All the usual type formers can be defined by coinduction.

Given $A : \mathcal{U}$ and $B : \mathcal{U}$, we define $A \times B : \mathcal{U}$ by

1. The elements of $A \times B$ are pairs (a, b) where $a : A$ and $b : B$.
2. $\mathsf{Id}_{A\times B}((a_1, b_1), (a_2, b_2)) = \mathsf{Id}_A(a_1, a_2) \times \mathsf{Id}_B(b_1, b_2)$.

The second is like "tail(ones) = ones": it defines the corecursive eliminator Id to equal (another instance of) the object being defined.

Type formers, coinductively

All the usual type formers can be defined by coinduction.

Given $A : \mathcal{U}$ and $B : \mathcal{U}$, we define $A \times B : \mathcal{U}$ by

1. The elements of $A \times B$ are pairs (a, b) where $a : A$ and $b : B$.
2. $\mathsf{Id}_{A\times B}((a_1, b_1), (a_2, b_2)) = \mathsf{Id}_A(a_1, a_2) \times \mathsf{Id}_B(b_1, b_2)$.

The second is like "tail(ones) = ones": it defines the corecursive eliminator Id to equal (another instance of) the object being defined.

Given $A : \mathcal{U}$ and $B : \mathcal{U}$, we define $A \to B : \mathcal{U}$ by

1. The elements of $A \to B$ are functions from A to B.
2. $\mathsf{Id}_{A\to B}(f, g) = (x : A) \to \mathsf{Id}_B(f(x), g(x))$??

This doesn't quite work: $(x : A) \to \mathsf{Id}_B(f(x), g(x))$ is not an instance of the same type former $A \to B$. We need to "generalize the coinductive hypothesis".

Type formers, coinductively

All the usual type formers can be defined by coinduction.

Given $A : \mathcal{U}$ and $B : \mathcal{U}$, we define $A \times B : \mathcal{U}$ by

1. The elements of $A \times B$ are pairs (a, b) where $a : A$ and $b : B$.
2. $\mathsf{Id}_{A\times B}((a_1, b_1), (a_2, b_2)) = \mathsf{Id}_A(a_1, a_2) \times \mathsf{Id}_B(b_1, b_2)$.

The second is like "tail(ones) = ones": it defines the corecursive eliminator Id to equal (another instance of) the object being defined.

Given $A : \mathcal{U}$ and $B : A \to \mathcal{U}$, we define $(x : A) \to B(x) : \mathcal{U}$ by

1. The elements of $(x : A) \to B$ are dependent functions from A to B, sending each $x : A$ to an element of $B(x)$.
2. $\mathsf{Id}_{(x:A)\to B(x)}(f, g) = (x : A) \to \mathsf{Id}_{B(x)}(f(x), g(x))$

(Not quite right for a different reason; we'll come back to it later.)

Coinductive universes are open

A coinductive universe is an open universe.

- We don't have to list type formers when we define the universe.
- If we invent or discover a new type former, we can just define a new element of $\mathcal{U}$ by corecursion.

In addition:

- Defining a type includes defining its identity types.
  (Defining a category/groupoid includes defining its morphisms.)

Coinductive universes are univalent

Define each universe, in the next higher universe, as $\mathcal{U} : \mathcal{U}'$ where

1. The elements of $\mathcal{U}$ are (small) types.
2. $\mathrm{Id}_{\mathcal{U}}(A, B) = (A \cong B)$.

This is the basic intuition of Higher Observational Type Theory.
But there are a lot of details to make it precise mathematically.

NB: As with LOTT, universes in HOTT are not coinductively defined inside the theory; this is only metatheoretic philosophy/intuition.
Later, we will make it partially precise inside another theory "POTT".

Outline

What makes equality equality?

I rather glibly wrote equations like

$$\mathrm{Id}_{A\times B}((a_1, b_1), (a_2, b_2)) = \mathrm{Id}_A(a_1, a_2) \times \mathrm{Id}_B(b_1, b_2)$$

but this only defines the type $\mathrm{Id}_{A\times B}$.

Equality is not just a type (or proposition), but comes with laws:

- Reflexivity: $x = x$ for all x.
- Symmetry: if $x = y$, then $y = x$.
- Transitivity: if $x = y$ and $y = z$, then $x = z$.
- Congruence: if $x = y$, then $f(x) = f(y)$.
- Substitution: if $x = y$ and $P(x)$, then $P(y)$.

What makes equality equality?

I rather glibly wrote equations like

$$\mathrm{Id}_{A\times B}((a_1, b_1), (a_2, b_2)) = \mathrm{Id}_A(a_1, a_2) \times \mathrm{Id}_B(b_1, b_2)$$

but this only defines the type $\mathrm{Id}_{A\times B}$.

Equality is not just a type (or proposition), but comes with laws:

- Reflexivity: $x = x$ for all x.
- Symmetry: if $x = y$, then $y = x$.
- Transitivity: if $x = y$ and $y = z$, then $x = z$.
- Congruence: if $x = y$, then $f(x) = f(y)$.
- Substitution: if $x = y$ and $P(x)$, then $P(y)$.

To start with we will focus on just two: reflexivity and congruence.

Reflexivity = congruence

In fact, reflexivity and congruence are two sides of the same coin.

1. Need not just unary congruence "if $x = y$, then $f(x) = f(y)$", but
   - Binary congruence: if $x = y$ and $u = v$, then $f(x, u) = f(y, v)$.
   - Ternary congruence: if $x = y$ and $u = v$ and $w = z$, then $f(x, u, w) = f(y, v, z)$.
   - etc.

   Reflexivity is just nullary congruence: if (nothing), then $f = f$.
2. With a better choice of $\mathrm{Id}_{A\to B}$, we get congruence from reflexivity:

$$\begin{aligned}\mathrm{Id}_{A\to B}(f,g) &= (x_0 : A)(x_1 : A)(x_2 : \mathrm{Id}_A(x_0, x_1)) \to \mathrm{Id}_B(f(x_0), g(x_1))\\ \mathrm{refl}_f \;:\; &\mathrm{Id}_{A\to B}(f,f)\\ &= (x_0 : A)(x_1 : A)(x_2 : \mathrm{Id}_A(x_0, x_1)) \to \mathrm{Id}_B(f(x_0), f(x_1))\end{aligned}$$

"Two functions are equal if they map equal inputs to equal outputs.
Since one function is equal to itself, it maps equal inputs to equal outputs."

Defining reflexivity

We define reflexivity/congruence term-by-term for both introduction and elimination forms.

For $A \times B$, recall $\mathsf{Id}_{A\times B}((a_1, b_1), (a_2, b_2)) = \mathsf{Id}_A(a_1, a_2) \times \mathsf{Id}_B(b_1, b_2)$.

- Introduction is pairs (a, b):

$$\mathsf{refl}_{(a,b)} = (\mathsf{refl}_a, \mathsf{refl}_b)$$

- Elimination is projections $\pi_1 : A \times B \to A$ and $\pi_2 : A \times B \to B$:

$$\mathsf{refl}_{\pi_1(u)} = \pi_1(\mathsf{refl}_u)$$
$$\mathsf{refl}_{\pi_2(u)} = \pi_2(\mathsf{refl}_u)$$

We consider this part of the definition of the "elements" of a type.

Two missing pieces that fit together

1. The "better choice of equality" for dependent functions:

$$\mathsf{Id}_{(x:A)\to B(x)}(f,g)$$
$$= (x_0 : A)(x_1 : A)(x_2 : \mathsf{Id}_A(x_0, x_1)) \to \mathsf{Id}_?(f(x_0), g(x_1))$$

needs a "heterogeneous equality" between $f(x_0) : B(x_0)$ and $g(x_1) : B(x_1)$ (of different types).

Two missing pieces that fit together

1. The "better choice of equality" for dependent functions:

$$\mathsf{Id}_{(x:A)\to B(x)}(f,g)$$
$$= (x_0 : A)(x_1 : A)(x_2 : \mathsf{Id}_A(x_0,x_1)) \to \mathsf{Id}_?(f(x_0), g(x_1))$$

   needs a "heterogeneous equality" between $f(x_0) : B(x_0)$ and $g(x_1) : B(x_1)$ (of different types).

2. Also, given $B : A \to \mathcal{U}$ and $x_2 : \mathsf{Id}_A(x_0,x_1)$, by congruence

$$\mathsf{refl}_B(x_2) : \mathsf{Id}_{\mathcal{U}}(B(x_0), B(x_1)).$$

Two missing pieces that fit together

1. The "better choice of equality" for dependent functions:

$$\mathsf{Id}_{(x:A)\to B(x)}(f,g) \\ = (x_0 : A)(x_1 : A)(x_2 : \mathsf{Id}_A(x_0,x_1)) \to \mathsf{Id}_?(f(x_0),g(x_1))$$

   needs a "heterogeneous equality" between $f(x_0) : B(x_0)$ and $g(x_1) : B(x_1)$ (of different types).

2. Also, given $B : A \to \mathcal{U}$ and $x_2 : \mathsf{Id}_A(x_0,x_1)$, by congruence

$$\mathsf{refl}_B(x_2) : \mathsf{Id}_{\mathcal{U}}(B(x_0),B(x_1)).$$

We stipulate every equality $C_2 : \mathsf{Id}_{\mathcal{U}}(C_0,C_1)$ induces a type family $\lfloor C_2 \rfloor : C_0 \times C_1 \to \mathcal{U}$. Then $\mathsf{Id}_{(x:A)\to B(x)}(f,g)$ can be

$$(x_0 : A)(x_1 : A)(x_2 : \mathsf{Id}_A(x_0,x_1)) \to \lfloor \mathsf{refl}_B(x_2) \rfloor (f(x_0),g(x_1)).$$

Think of $C_2 : \mathsf{Id}_{\mathcal{U}}(C_0,C_1)$ as a one-to-one correspondence, where $\lfloor C_2 \rfloor (x,y)$ says that $x : C_0$ and $y : C_1$ correspond.

Parametricity

If we stop here, with $\mathsf{Id}_{\mathcal{U}}$ nothing but these correspondences:

$$\mathsf{Id}_{\mathcal{U}}(A, B) = (A \times B \to \mathcal{U})$$

we get a Parametric-Observational Type Theory (POTT).

- Id_A is called a "bridge type" rather than an "equality type"
- Has semantics in semicartesian ("BCH") cubical sets
- Fully implemented in the experimental proof assistant `narya -parametric`
- Executes as a programming language (canonicity proven, normalization implemented and conjectured correct)
- Other internally parametric type theories are cubical-style, with all its drawbacks

  ...and, as far as I know, none of them are implemented.

(I omitted some details, like the "transposition" operator on cubes.)

Outline

What's left

We are still missing:

- Symmetry: $x = y$ implies $y = x$.
- Transitivity: $x = y$ and $y = z$ imply $x = z$.
- Substitution: $x = y$ and $P(x)$ imply $P(y)$.
- One-to-one-ness of the correspondence $\lfloor C_2 \rfloor$.

Adding these to POTT will produce HOTT.

At this step we can actually work inside the theory POTT to construct a model of HOTT and deduce its rules.

Substitution

We will focus on substitution, since it implies the others:

- If $x = y$, let $P(u) = (u = x)$.
  Then $P(x)$, so $P(y)$, meaning $y = x$.

- If $x = y$ and $y = z$, let $P(u) = (x = u)$.
  Then $P(y)$ and $y = z$, so $P(z)$, meaning $x = z$.

- If $C_0 = C_1$ in $\mathcal{U}$, let $P(X) = X$.
  Then if $x : C_0$, also $x : P(C_0)$, so we get something in $P(C_1)$, that is, in C_1. Similarly, we get $C_1 \to C_0$, etc...

Fibrancy

We think of substitution:

$$\text{if } P : A \to \mathcal{U} \text{ and } a_2 : \mathrm{Id}_A(a_0, a_1), \text{ then } P(a_0) \to P(a_1)$$

as a property of P. Homotopically/categorically, it is path lifting.

We want to define a universe $\mathcal{U}_{\mathrm{fib}} = (A : \mathcal{U}) \times \mathrm{isFibrant}(A)$ of fibrant types, such that any $P : A \to \mathcal{U}_{\mathrm{fib}}$ satisfies substitution. Taking $\mathcal{U}_{\mathrm{fib}}$ as "the" universe will yield HOTT.

This is similar to other situations:

- Building models of cubical type theory in "Orton-Pitts" style
- Defining fibration classifiers in synthetic ∞-category theory

As there, we rely on the "amazing right adjoint". But instead of axiomatizing it with modalities, we use higher coinductive types.

Higher coinductive types

"Recall": a higher inductive type contains constructors whose output lies in an identity type rather than the type itself.

- The circle S^1 is constructed by $b : S^1$ and $\ell : \mathsf{Id}_{S^1}(b, b)$.
- Truncation $\|A\|$ is constructed by $p : A \to \|A\|$ and $q : (x, y : \|A\|) \to \mathsf{Id}_{\|A\|}(x, y)$ (a recursive constructor).

Higher coinductive types

"Recall": a higher inductive type contains constructors whose output lies in an identity type rather than the type itself.

- The circle S^1 is constructed by $b : S^1$ and $\ell : \mathsf{Id}_{S^1}(b, b)$.
- Truncation $\|A\|$ is constructed by $p : A \to \|A\|$ and $q : (x, y : \|A\|) \to \mathsf{Id}_{\|A\|}(x, y)$ (a recursive constructor).

Dually, a higher coinductive type contains eliminators whose input lies in an identity type rather than the type itself.

- The amazing right adjoint $\sqrt{A}$ is eliminated by

$$\mathsf{root} : (x_0, x_1 : \sqrt{A})(x_2 : \mathsf{Id}_{\sqrt{A}}(x_0, x_1)) \to A.$$

(You may never have contemplated such a thing before.
It's a good thing the universe is open!)

Higher coinductive fibrancy

We define isFibrant(Q), for $Q : \mathcal{U}$, to have five "higher eliminators". Given $Q_2 : \mathrm{Id}_{\mathcal{U}}(Q_0, Q_1)$ and $\phi_2 : \lfloor \mathrm{refl}_{\mathsf{isFibrant}}(Q_2) \rfloor(\phi_0, \phi_1)$, hence $\phi_0 : \mathsf{isFibrant}(Q_0)$ and $\phi_1 : \mathsf{isFibrant}(Q_1)$, the eliminators yield:

$$
\begin{aligned}
\mathsf{trr} &: Q_0 \to Q_1 \\
\mathsf{liftr} &: (x_0 : Q_0) \to Q_2(x_0, \mathsf{trr}(x_0)) \\
\mathsf{trl} &: Q_1 \to Q_0 \\
\mathsf{liftl} &: (x_1 : Q_1) \to Q_2(\mathsf{trl}(x_1), x_1) \\
\mathsf{id} &: (x_0 : Q_0)(x_1 : Q_1) \to \mathsf{isFibrant}(Q_2(x_0, x_1))
\end{aligned}
$$

Intuitively, trr and trl say Q admits substitution "over any base", in the following sense...

From fibrancy to substitution

Suppose $P : A \to \mathcal{U}$ and $\phi : (a : A) \to \mathsf{isFibrant}(P(a))$, in other words $(P, \phi) : A \to \mathcal{U}_{\mathsf{fib}}$.

Then for any $a_2 : \mathsf{Id}_A(a_0, a_1)$ we have

$$\mathsf{refl}_\phi(a_2) : \lfloor \mathsf{refl}_{\mathsf{isFibrant}}(\mathsf{refl}_P(a_2)) \rfloor (\phi(a_0), \phi(a_1)).$$

Thus $\mathsf{trr} : P(a_0) \to P(a_1)$ and $\mathsf{trl} : P(a_1) \to P(a_0)$, so any family of fibrant types has substitution.

Also, $\mathsf{liftr} : (x_0 : P(a_0)) \to \lfloor \mathsf{refl}_P(a_2) \rfloor (x_0, \mathsf{trr}(x_0))$ and dually: a substituted element is heterogeneously equal to the original.

Finally, the last eliminator "id" says that $\mathcal{U}_{\mathsf{fib}}$ is closed under Id.

Fibrant type formers

We can now prove, in POTT, that all the type formers are fibrant, by "higher coinduction".

For example, we define $\mathsf{fibProd}(A, B) : \mathsf{isFibrant}(A \times B)$ by:

$$\begin{aligned}
\mathsf{trr}((a_0, b_0)) &= (\mathsf{trr}(a_0), \mathsf{trr}(b_0)) \\
\mathsf{liftr}((a_0, b_0)) &= (\mathsf{liftr}(a_0), \mathsf{liftr}(b_0)) \\
\mathsf{trl}((a_1, b_1)) &= (\mathsf{trl}(a_1), \mathsf{trl}(b_1)) \\
\mathsf{liftl}((a_1, b_1)) &= (\mathsf{liftl}(a_1), \mathsf{liftl}(b_1)) \\
\mathsf{id}((a_0, b_0), (a_1, b_1)) &= \mathsf{fibProd}(\mathsf{Id}_A(a_0, a_1), \mathsf{Id}_B(b_0, b_1)).
\end{aligned}$$

For $A \to B$ we have $\mathsf{trr}(f_0) = (a_1 \mapsto \mathsf{trr}_B(f_0(\mathsf{trl}_A(a_1))))$, etc.

(Fibrancy of $\mathcal{U}_{\mathsf{fib}}$ itself is the hardest case; this is work in progress.)

Higher structure

Theorem

Fibrant types admit the Martin-Löf eliminator J for their identity types, with a propositional computation rule.

Thus, all fibrant types have the structure of an ∞-groupoid, as in Book HoTT and cubical type theory.

Univalence

Theorem

$\mathcal{U}_{\mathsf{fib}}$ satisfies the univalence principle.

Specifically, $\mathsf{Id}_{\mathcal{U}_{\mathsf{fib}}}(A, B)$ is equivalent to the type of bisimulations.
A bisimulation is a correspondence $E : A \times B \to \mathcal{U}_{\mathsf{fib}}$ such that

- For any $a : A$ there is a $b : B$ and $e : E(a, b)$.
- For any $b : B$ there is an $a : A$ and $e : E(a, b)$.
- For any $e_0 : E(a_0, b_0)$ and $e_1 : E(a_1, b_1)$, the correspondence*

$$\lfloor \mathsf{refl}_E(-,-) \rfloor : \mathsf{Id}_A(a_0, a_1) \times \mathsf{Id}_B(b_0, b_1) \to \mathcal{U}_{\mathsf{fib}}$$

  is a bisimulation, coinductively.

These are equivalent to any other good notion of equivalence.

These theorems are fully formalized in Narya.

* Actually this is a higher destructor, but the basic idea is the same.

Higher Observational Type Theory

Since $\mathcal{U}_{\mathsf{fib}}$ is closed under all type formers, working in POTT we have built another model of type theory where $\mathcal{U}_{\mathsf{fib}}$ is the universe. This new model satisfies:

- All the rules of POTT except $\mathsf{Id}_{\mathcal{U}}(A, B) = (A \times B \to \mathcal{U})$.
- Instead, it satisfies univalence $\mathsf{Id}_{\mathcal{U}}(A, B) = (A \cong B)$.
- And "all types are fibrant": if $A_2 : \mathsf{Id}_{\mathcal{U}}(A_0, A_1)$, we have

$$\mathsf{trr} : A_0 \to A_1 \qquad \mathsf{liftr} : (x_0 : A_0) \to A_2(x_0, \mathsf{trr}(x_0))$$
$$\mathsf{trl} : A_1 \to A_0 \qquad \mathsf{liftl} : (x_1 : A_1) \to A_2(\mathsf{trl}(x_1), x_1).$$

  which compute on all the type formers in the way we defined them to when proving fibrancy in POTT.

POTT with these changes is Higher Observational Type Theory. It is implemented in `narya` (default mode), except for computing fibrancy on inductive types and the universe.

Why HOTT?

- Conjecturally, a computational implementation of univalence (as cubical type theory is known to be).
- Does not build ∞-groupoids in explicitly. Higher structure is emergent from a few simple rules, as in Book HoTT.
- Philosophically justified by harmony and a coinductive universe.
- Univalence is "definitional" (stronger than cubical type theory).
- Structure identity principles are also definitional.
- No interval variables: easier to use, and (eventually) easier to implement important features like unification.

`http://narya.readthedocs.io`