Observational Equality

Now for Good

Martin-Löf Type Theory is Awesome!

MLTT is a jewel of the Curry-Howard correspondence.

> Expressive enough to do a lot of mathematics

> Powerful enough to define most computable functions

> Decidable

Your computer can always tell whether your proof is correct or not

> Normalization and canonicity

You don’t need a lemma to prove that foo(7) is 42. It always computes!

The Inductive Equality is not Awesome

```
Inductive eq (A : Type) (a : A) : A -> Type :=
| eq_refl : eq A a a
```

The equality supplied by MLTT encodes equality of programs, not equality of behaviours.

Canonicity → in the empty context, the only equality proof is **`eq_refl`**, which means the terms have to be convertible.

Decidability → the inductive equality is decidable.

The Inductive Equality is not Awesome

```
Inductive eq (A : Type) (a : A) : A -> Type :=
| eq_refl : eq A a a
```

Two unpleasant consequences:

> no function extensionality

You can prove that for all n, n+1 = 1+n
You cannot prove that λn . n+1 = λn . 1+n

> no quotient types

Given a relation R on a type A, you cannot form the quotient A/R

Possible workarounds

> Use axioms : just postulate function extensionality, etc

> Use setoids : equip every type with an equivalence relation, and ensure that functions preserve them.

> Add the reflection rule for equality (extensional type theory)

> Use cubical type theory

Observational Type Theory

Altenkirch and McBride designed OTT to fix the inductive equality.

Main insight: instead of being an inductive data structure, equality is defined by recursion on the types

$$2 \sim_{\mathbb{N}} 2$$

Observational Type Theory

Altenkirch and McBride designed OTT to fix the inductive equality.

Main insight: instead of being an inductive data structure, equality is defined by recursion on the types

$$2 \sim_{\boxed{\mathbb{N}}} 2$$

Observational Type Theory

Altenkirch and McBride designed OTT to fix the inductive equality.

Main insight: instead of being an inductive data structure, equality is defined by recursion on the types

$$2 \sim_{\mathbb{N}} 2 \longrightarrow 1 \sim_{\mathbb{N}} 1$$

Observational Type Theory

Altenkirch and McBride designed OTT to fix the inductive equality.

Main insight: instead of being an inductive data structure, equality is defined by recursion on the types

$$2 \sim_{\mathbb{N}} 2 \longrightarrow 1 \sim_{\mathbb{N}} 1 \longrightarrow 0 \sim_{\mathbb{N}} 0$$

Observational Type Theory

Altenkirch and McBride designed OTT to fix the inductive equality.

Main insight: instead of being an inductive data structure, equality is defined by recursion on the types

$$2 \sim_{\mathbb{N}} 2 \longrightarrow 1 \sim_{\mathbb{N}} 1 \longrightarrow 0 \sim_{\mathbb{N}} 0 \longrightarrow \top$$

Observational Type Theory

Altenkirch and McBride designed OTT to fix the inductive equality.

Main insight: instead of being an inductive data structure, equality is defined by recursion on the types

$$2 \sim_{\mathbb{N}} 2 \longrightarrow 1 \sim_{\mathbb{N}} 1 \longrightarrow 0 \sim_{\mathbb{N}} 0 \longrightarrow \top$$

$$f \sim_{A \to B} g$$

Observational Type Theory

Altenkirch and McBride designed OTT to fix the inductive equality.

Main insight: instead of being an inductive data structure, equality is defined by recursion on the types

$$2 \sim_{\mathbb{N}} 2 \longrightarrow 1 \sim_{\mathbb{N}} 1 \longrightarrow 0 \sim_{\mathbb{N}} 0 \longrightarrow \top$$

$$f \sim_{\boxed{A \to B}} g$$

Observational Type Theory

Altenkirch and McBride designed OTT to fix the inductive equality.

Main insight: instead of being an inductive data structure, equality is defined by recursion on the types

$$2 \sim_{\mathbb{N}} 2 \longrightarrow 1 \sim_{\mathbb{N}} 1 \longrightarrow 0 \sim_{\mathbb{N}} 0 \longrightarrow \top$$

$$f \sim_{A \to B} g \longrightarrow \prod_{x:A} f\, x \sim_B g\, x$$

TT^{obs} : *Yet Another Flavor of OTT*

Eliminating observational equality

$$\frac{P : \mathbb{N} \to \mathsf{Type} \qquad n, m : \mathbb{N} \qquad e : m \sim_{\mathbb{N}} n \qquad t : P\, m}{P\, n}$$

Eliminating observational equality

$$\frac{P : \mathbb{N} \to \mathsf{Type} \qquad n, m : \mathbb{N} \qquad e : m \sim_{\mathbb{N}} n \qquad t : P\, m}{P\, n}$$

$$\frac{A, B : \mathsf{Type} \qquad e : A \sim_{\mathsf{Type}} B \qquad x : A}{\mathsf{cast}(A, B, e, x) : B}$$

Eliminating observational equality

$$\frac{P : \mathbb{N} \to \mathsf{Type} \qquad n, m : \mathbb{N} \qquad e : m \sim_{\mathbb{N}} n \qquad t : P\, m}{\mathsf{cast}(P\, m, P\, n, \mathsf{ap}_f\, e, t) : P\, n}$$

$$\frac{A, B : \mathsf{Type} \qquad e : A \sim_{\mathsf{Type}} B \qquad x : A}{\mathsf{cast}(A, B, e, x) : B}$$

Eliminating observational equality

As with observational equality, it computes by recursion on types and terms:

$$\mathsf{cast}(A \to B, A' \to B', e, f)$$

Eliminating observational equality

As with observational equality, it computes by recursion on types and terms:

$$\mathsf{cast}(\underline{A \to B}, \underline{A' \to B'}, e, f)$$

compatible

Eliminating observational equality

As with observational equality, it computes by recursion on types and terms:

$$\mathsf{cast}(A \to B, A' \to B', e, f) \longrightarrow$$

$$\lambda(x : A').\,\mathsf{cast}(B, B', \pi_2\, e, f\, \mathsf{cast}(A', A, \pi_1\, e^{-1}, x))$$

Eliminating observational equality

As with observational equality, it computes by recursion on types and terms:

$$\mathsf{cast}(A \to B, A' \to B', e, f) \longrightarrow$$

$$\lambda(x : A').\, \mathsf{cast}(B, B', \pi_2\, e, f\, \mathsf{cast}(A', A, \pi_1\, e^{-1}, x))$$

$$e : (A \to B) \sim_{\mathsf{Type}} (A' \to B')$$

Eliminating observational equality

As with observational equality, it computes by recursion on types and terms:

$$\mathsf{cast}(A \to B, A' \to B', e, f) \longrightarrow$$

$$\lambda(x : A').\,\mathsf{cast}(B, B', \pi_2\, e, f\, \mathsf{cast}(A', A, \pi_1\, e^{-1}, x))$$

$$e : (A \sim_{\mathsf{Type}} A') \times (B \sim_{\mathsf{Type}} B')$$

Definitional Proof-Irrelevance

How do we prove reflexivity or transitivity of the equality with cast?

We can't!

Definitional Proof-Irrelevance

How do we prove reflexivity or transitivity of the equality with cast?

We can't!

Second insight of OTT: we need a layer of *proof-irrelevant* types that will contain the observational equality.

Now any two proofs of the same equality are undistinguishable
→ definitional K/UIP

Inductive Types

Regular inductive types work just fine.

However, indexed inductive types need a new constructor to handle cast values, which might not have a canonical form.

For instance, the inductive equality becomes:

```
Inductive eq (A : Type) (a : A) : A -> Type :=
| eq_refl : eq A a a
| eq_cast : forall b, a ~A b -> eq A a b
```

This is the OTT analogue to Swan's encoding of equality types.
It implies that canonicity is weakened for indexed inductive types.

But wait, there's more!

These three insights make TT^{obs} into a proper extension of MLTT (all MLTT proofs remain valid!) that adds extensionality principles.

But we can add more:

> quotients of a type by a *proof-irrelevant* equivalence relation

> irrelevant squash types

> subset types

Quotient types

$$\frac{A : \mathsf{Type} \qquad R : A \to A \to \mathsf{Prop} \qquad \mathsf{equiv}(R)}{A/R : \mathsf{Type}}$$

$$\pi_{A/R} : A \to A/R$$

$$\pi_{A/R}\, x \sim_{A/R} \pi_{A/R}\, y \longrightarrow R\ x\ y$$

Meta-Theory

So far, these ideas are not particularly new (they can actually be pieced together from McBride's papers and blog posts).

Our main contribution is a proper development of the meta-theory of TT^{obs}, to prove normalization, canonicity and decidability of type-checking.

Consistency

Consistency can be proved by constructing a model. This can be done in a constructive set theory (or a type theory) that is strong enough to do induction-recursion, or plain ZF set theory.

From there, we obtain that

> there are no inhabitants of $\bot$ in the empty context

> there are no proofs of anti-diagonal equalities between types

Normalization and canonicity

Normalization, canonicity and decidability of conversion can be proved using logical relations.

We used the induction-recursion based framework of Abel, Öhman and Vezzosi to formally prove these three properties in Agda.

Normalization and canonicity

> No computation in Prop → all terms are reducible as long as they are well-typed, and we can have arbitrary axioms. Which means we lose any control on which propositions are inhabited!

> As a consequence, the logical relation does not allow us to prove there are no neutral terms in the empty context. But we can get this from the model.

> Reducibility of cast relies on having an inductive description of the inhabitants of Type. This does not seem compatible with reducibility candidates, which are the usual way to handle impredicativity...

Implementation is not too Difficult

All in all, we only need three ingredients:

> Definitionally proof-irrelevant types

Already featured in Coq, Agda and Lean

> Two primitives cast and ~, along with rewriting rules

> A new constructor for indexed inductive types

We used Jesper Cockx's rewrite rules to implement

TT^{obs} in Agda.

There are plans to add it as an option to the Coq kernel

Thank you