Choice Principles in Observational Type Theory

Loïc Pujet

March 8, 2024

1. Observational Type Theory

A Type Theory for Set Truncated Mathematics

What should we have in a type theory for classical set-truncated math?

According to the Lean proof assistant,

A Type Theory for Set Truncated Mathematics

What should we have in a type theory for classical set-truncated math?
According to the Lean proof assistant,

- A sort of proposition SProp with
  - definitional proof-irrelevance

A Type Theory for Set Truncated Mathematics

What should we have in a type theory for classical set-truncated math?
According to the Lean proof assistant,

- A sort of proposition SProp with
  - definitional proof-irrelevance

$$\frac{P : SProp \qquad a : P \qquad b : P}{a \equiv b : P}$$

A Type Theory for Set Truncated Mathematics

What should we have in a type theory for classical set-truncated math?
According to the Lean proof assistant,

- A sort of proposition SProp with
  - definitional proof-irrelevance
  - impredicativity

A Type Theory for Set Truncated Mathematics

What should we have in a type theory for classical set-truncated math?
According to the Lean proof assistant,

- A sort of proposition SProp with
  - definitional proof-irrelevance
  - impredicativity

$$\frac{\Gamma, A \vdash B : SProp}{\Gamma \vdash \Pi A B : SProp}$$

A Type Theory for Set Truncated Mathematics

What should we have in a type theory for classical set-truncated math?
According to the Lean proof assistant,

- A sort of proposition SProp with
  - definitional proof-irrelevance
  - impredicativity
  - large elimination of sub-singletons inductive types

A Type Theory for Set Truncated Mathematics

What should we have in a type theory for classical set-truncated math?
According to the Lean proof assistant,

- A sort of proposition SProp with
  - definitional proof-irrelevance
  - impredicativity
  - large elimination of sub-singletons inductive types

False

A Type Theory for Set Truncated Mathematics

What should we have in a type theory for classical set-truncated math?
According to the Lean proof assistant,

- A sort of proposition SProp with
  - definitional proof-irrelevance
  - impredicativity
  - large elimination of sub-singletons inductive types

False Equality

A Type Theory for Set Truncated Mathematics

What should we have in a type theory for classical set-truncated math?
According to the Lean proof assistant,

- A sort of proposition SProp with
  - definitional proof-irrelevance
  - impredicativity
  - large elimination of sub-singletons inductive types

False   Equality   Accessibility

A Type Theory for Set Truncated Mathematics

What should we have in a type theory for classical set-truncated math?
According to the Lean proof assistant,

- A sort of proposition SProp with
  - definitional proof-irrelevance
  - impredicativity
  - large elimination of sub-singletons inductive types
- Quotient types

A Type Theory for Set Truncated Mathematics

What should we have in a type theory for classical set-truncated math?
According to the Lean proof assistant,

- A sort of proposition SProp with
  - definitional proof-irrelevance
  - impredicativity
  - large elimination of sub-singletons inductive types
- Quotient types

$$\frac{A : Type_{\ell} \qquad R : A \to A \to SProp}{A/R : Type_{\ell}}$$

+ canonical projection, elimination rule, computation rule

A Type Theory for Set Truncated Mathematics

What should we have in a type theory for classical set-truncated math? According to the Lean proof assistant,

- A sort of proposition SProp with
  - definitional proof-irrelevance
  - impredicativity
  - large elimination of sub-singletons inductive types
- Quotient types
- Function extensionality & Proposition extensionality

A Type Theory for Set Truncated Mathematics

What should we have in a type theory for classical set-truncated math?
According to the Lean proof assistant,

- A sort of proposition SProp with
  - definitional proof-irrelevance
  - impredicativity
  - large elimination of sub-singletons inductive types
- Quotient types
- Function extensionality & Proposition extensionality
- Excluded middle & Choice (Hilbert's ϵ operator)

Mathematics vs Type Theory

Unfortunately, these combined features break some important properties of type theory:

Large elimination of Accessibility + definitional proof irrelevance makes type-checking undecidable and breaks subject reduction.

The computation rule for Lean's quotients in SProp + definitional proof irrelevance makes type-checking undecidable and breaks subject reduction.

The computation rule for Lean's J + computation in impredicative types makes type-checking undecidable. [Abel and Coquand 2019]

Observational Type Theory

Observational Type Theory [Altenkirch, McBride, Swierstra 2007] is an internal language for types equipped with a proof-irrelevant equality relation (proof-irrelevant setoids).

Observational Type Theory

Observational Type Theory [Altenkirch, McBride, Swierstra 2007] is an internal language for types equipped with a proof-irrelevant equality relation (proof-irrelevant setoids).

The central notion in OTT is the observational equality

$$\frac{A : Type_{\ell} \qquad B : Type_{\ell} \qquad a : A \qquad b : B}{a \;{}_{A}{\sim}_{B}\; b : SProp}$$

Observational Type Theory

Observational Type Theory [Altenkirch, McBride, Swierstra 2007] is an internal language for types equipped with a proof-irrelevant equality relation (proof-irrelevant setoids).

The central notion in OTT is the observational equality

$$\frac{A : Type_\ell \qquad B : Type_\ell \qquad a : A \qquad b : B}{a\ {}_{A}{\sim}_{B}\ b : SProp}$$

This heterogeneous, proof-irrelevant relation replaces the usual Martin-Löf identity type.

Observational Type Theory

The observational equality is defined on a type-per-type basis:

Observational Type Theory

The observational equality is defined on a type-per-type basis:

- an equality between two pairs is a pair of equalities

$$t \ {}_{A\times B}{\sim}_{C\times D} \ u \quad \equiv \quad (fst\ t\ {}_{A}{\sim}_{C}\ fst\ u) \wedge (snd\ t\ {}_{B}{\sim}_{D}\ snd\ u)$$

Observational Type Theory

The observational equality is defined on a type-per-type basis:

- an equality between two pairs is a pair of equalities

$$t \;{}_{A\times B}\!\sim_{C\times D}\; u \quad \equiv \quad (fst\; t\; {}_{A}\!\sim_{C}\; fst\; u) \wedge (snd\; t\; {}_{B}\!\sim_{D}\; snd\; u)$$

- an equality between two functions is a family of pointwise equality

$$f \;{}_{\Pi AB}\!\sim_{\Pi CD}\; g \quad \equiv \quad \Pi(x : A)(y : C)\,.\, x\; {}_{A}\!\sim_{C}\; y \;\rightarrow\; f\,x\; {}_{B[x]}\!\sim_{D[y]}\; g\,y$$

Observational Type Theory

The observational equality is defined on a type-per-type basis:

- an equality between two pairs is a pair of equalities

$$t \;{}_{A\times B}{\sim}_{C\times D}\; u \quad \equiv \quad (fst\, t \;{}_{A}{\sim}_{C}\; fst\, u) \wedge (snd\, t \;{}_{B}{\sim}_{D}\; snd\, u)$$

- an equality between two functions is a family of pointwise equality

$$f \;{}_{\Pi AB}{\sim}_{\Pi CD}\; g \quad \equiv \quad \Pi(x : A)(y : C)\,.\, x \;{}_{A}{\sim}_{C}\; y \;\rightarrow\; f\, x \;{}_{B[x]}{\sim}_{D[y]}\; g\, y$$

- an equality across two incompatible types is false

$$t \;{}_{\Pi AB}{\sim}_{\mathbb{N}}\; u \quad \equiv \quad \bot$$

Observational Type Theory

Most of the properties of equality are postulated as proof irrelevant axioms.

- reflexivity
- symmetry
- transitivity
- function congruence
- etc...

Observational Type Theory

To eliminate the observational equality, OTT provides a typecasting operator

$$\frac{A : Type_\ell \qquad B : Type_\ell \qquad e : A \sim B \qquad t : A}{cast(A, B, e, t) : B}$$

The cast operator computes by case analysis on A and B.

Observational Type Theory

To eliminate the observational equality, OTT provides a typecasting operator

$$\frac{A : Type_\ell \qquad B : Type_\ell \qquad e : A \sim B \qquad t : A}{cast(A, B, e, t) : B}$$

The cast operator computes by case analysis on A and B.

- casting between two product types is a component-wise cast

$$cast(A \times B, C \times D, e, t) \equiv \langle cast(A, C, e_1, fst\ t); cast(B, D, e_2, snd\ t)\rangle$$

Observational Type Theory

To eliminate the observational equality, OTT provides a typecasting operator

$$\frac{A : Type_{\ell} \qquad B : Type_{\ell} \qquad e : A \sim B \qquad t : A}{cast(A, B, e, t) : B}$$

The cast operator computes by case analysis on A and B.

- casting between two product types is a component-wise cast

$$cast(A \times B, C \times D, e, t) \equiv \langle cast(A, C, e_1, fst\ t); cast(B, D, e_2, snd\ t)\rangle$$

- casting between two function types is a back-and-forth cast

$$cast(A \to B, C \to D, e, f) \equiv \lambda x. cast(B, D, e_2, f\ cast(C, A, e_1, x))$$

Observational Type Theory

To eliminate the observational equality, OTT provides a typecasting operator

$$\frac{A : Type_\ell \qquad B : Type_\ell \qquad e : A \sim B \qquad t : A}{cast(A, B, e, t) : B}$$

The cast operator computes by case analysis on A and B.

Observational Type Theory

To eliminate the observational equality, OTT provides a typecasting operator

$$\frac{A : Type_\ell \qquad B : Type_\ell \qquad e : A \sim B \qquad t : A}{cast(A, B, e, t) : B}$$

The cast operator computes by case analysis on A and B.

On top of these computation rules, we add the rule cast-refl

$$cast(A, A, e, t) \equiv t$$

Observational Type Theory

We can define the usual J eliminator from cast and proof irrelevance.

$$\frac{A : Type \qquad x : A \qquad P : \Pi(z : A).x \;{}_{A}\!\sim_{A}\; z \to Type \qquad t : P\,x\,refl \qquad y : A \qquad e : x \;{}_{A}\!\sim_{A}\; y}{? : P\,y\,e}$$

Observational Type Theory

We can define the usual J eliminator from cast and proof irrelevance.

$$\frac{A : Type \qquad x : A \qquad P : \Pi(z : A).x \;{}_A{\sim}_A\; z \to Type \qquad t : P\; x\; refl \qquad y : A \qquad e : x \;{}_A{\sim}_A\; y}{cast(P\; x\; refl, P\; y\; e, ?, t) : P\; y\; e}$$

Observational Type Theory

We can define the usual J eliminator from cast and proof irrelevance.

$$\frac{A : Type \qquad x : A \qquad P : \Pi(z : A).x \;{}_{A}{\sim}_{A}\; z \to Type \qquad t : P\,x\,refl \qquad y : A \qquad e : x \;{}_{A}{\sim}_{A}\; y}{cast(P\,x\,refl, P\,y\,e, J_{SProp}(\lambda\, z\, e\,.\, P\,x\,refl \sim P\,z\,e, refl, y, e), t) : P\,y\,e}$$

Observational Type Theory

We can define the usual J eliminator from cast and proof irrelevance.

$$\frac{A : Type \qquad x : A \qquad P : \Pi(z : A).x \;{}_A{\sim}_A\; z \to Type \qquad t : P\,x\,refl \qquad y : A \qquad e : x \;{}_A{\sim}_A\; y}{cast(P\,x\,refl, P\,y\,e, J_{SProp}(\lambda\, z\, e\,.\, P\,x\,refl \sim P\,z\,e, refl, y, e), t) : P\,y\,e}$$

Thanks to the rule cast-refl, this J eliminator satisfies the usual computation rule.

$\to$ OTT is a superset of MLTT

Observational Type Theory

From these primitives, one obtains a theory that supports

- function and proposition extensionality
- definitional UIP
- impredicativity of SProp
- quotient types and their computation rule

Observational Type Theory

From these primitives, one obtains a theory that supports

- function and proposition extensionality
- definitional UIP
- impredicativity of SProp
- quotient types and their computation rule

Plus, OTT is type-theoretically well-behaved!

- consistency
- normalization
- subject reduction
- decidable conversion and type-checking

[Pujet and Tabareau 2023]

Coming soon to a proof assistant near you!

```
Set Observational Inductives.

Variable A B C : Set.
Variable obseq_list : list A ~ list B.
Variable a : A.

Eval lazy in (cast (list A) (list B) obseq_list (cons A a (nil A))).
U:---  *goals*        All (1,0)       (Coq Goals)
     = cons B (cast A B (obseq_cons_0 A B obseq_list) a) (nil B)
     : list B

U:%%-  *response*     All (1,0)       (Coq Response)
```

Coming soon to a proof assistant near you!

```
Set Observational Inductives.

Variable A B C : Set.
Variable obseq_list : list A ~ list B.
Variable a : A.

Eval lazy in (cast (list A) (list B) obseq_list (cons A a (nil A))).
U:---  *goals*        All (1,0)       (Coq Goals)
     = cons B (cast A B (obseq_cons_0 A B obseq_list) a) (nil B)
     : list B

U:%%-  *response*     All (1,0)       (Coq Response)
```

$$cast(list\ A, list\ B, e, [a]) \equiv [cast(A, B, e', a)]$$

(Implementation largely based on the work of Gilbert, Leray, Tabareau, Winterhalter)

2. Principles of Choice

Quotients in OTT

The rule for the formation of quotients ask for a SProp-valued relation

$$\frac{A : Type_\ell \qquad R : A \to A \to SProp}{A/R : Type_\ell}$$

Quotients in OTT

The rule for the formation of quotients ask for a SProp-valued relation

$$\frac{A : Type_\ell \qquad R : A \to A \to SProp}{A/R : Type_\ell}$$

Thus, if you have a relation $R : A \to A \to Type$, you need to quotient by the truncated relation $\|R\|$, where truncation is defined as an inductive type:

$$\begin{array}{l} Inductive \ \ \|\_\| \ \ (A : Type) \ : \ SProp := \\ |\_| : A \to \|A\| \end{array}$$

Quotients in OTT

Now, if you prove $\pi\, x \sim \pi\, y$ in the quotient type $A/\|R\|$, you can obtain a proof of $\|R\ x\ y\|$, but unfortunately not a proof of $R\ x\ y$.

"Quotients are not effective" [Sterling, Angiuli and Gratzer 2019]

In other words: once you transform a type into a proposition, it is really difficult to get back into the world of types.

Escaping truncation

Define a choice principle for A to be a function $\|A\| \to A$.

Escaping truncation

Define a choice principle for A to be a function $\|A\| \to A$.
To connect this to the usual definition of choice, note that if you define

$$\exists(x : A) . B \ := \ \|\Sigma(x : A) . B\|$$

Then a choice principle allows you to realise the familiar statement

$$\Pi(x : A)\exists(y : B) . R\, x\, y \to \exists(f : A \to B)\Pi(x : A) . R\, x\, (f\, x)$$

Escaping truncation

Define a choice principle for A to be a function $\|A\| \to A$.
To connect this to the usual definition of choice, note that if you define

$$\exists(x : A).B \;:=\; \|\Sigma(x : A).B\|$$

Then a choice principle allows you to realise the familiar statement

$$\Pi(x : A)\exists(y : B).R\,x\,y \to \exists(f : A \to B)\Pi(x : A).R\,x\,(f\,x)$$

Unfortunately, choice principles are uncommon in OTT: they basically exist for decidable types only

Choice Principles

Compare the situation with other type theories:

In Lean, the full axiom of choice taken as a postulate, thus you get a choice principle for all types.

Combined with extensionality principles, the full axiom of choice implies excluded middle, and is thus highly non-constructive.

Choice Principles

Compare the situation with other type theories:

In HoTT/CubicalTT, the role of propositions is played by hProps, and propositional truncation is defined as a HIT.

The eliminator of propositional truncation provides unique choice: if all inhabitants of P are equal, then you have a choice principle for P.

This is a sweet spot for constructive mathematics:

- quotients by hProp-valued equivalence relations are effective
- functions are identified with functional graphs

Choice Principles

Compare the situation with other type theories:

In Coq, you can implement some weaker choice principles using large elimination of the accessibility predicate.

In particular, you can show countable choice for decidable predicates: if P is a decidable predicate on $\mathbb{N}$, then

$$\exists (n : \mathbb{N}) \,.\, P\, n \quad \longrightarrow \quad \Sigma (n : \mathbb{N}) \,.\, P\, n$$

This is sufficient to define a lot of recursive functions by showing that their call graph is well-founded. Combined with impredicativity, this is enough to define an evaluator for System F.

Observational choice

Can we extend OTT with some choice principles?

Observational choice

Can we extend OTT with some choice principles?

We can not do much when propositions are proof-irrelevant: there is no information to extract from a proof of truncation $\|A\|$, meaning that a choice principle would have to invent an inhabitant of A out of thin air.

Observational choice

What if we give up proof-irrelevance? Then we could imagine

$$
\begin{aligned}
&choice : \|A\| \to isHProp(A) \to A \\
&choice\ |a|\ \_\ \equiv\ a
\end{aligned}
$$

Observational choice

What if we give up proof-irrelevance? Then we could imagine

$$\begin{aligned} &choice : \|A\| \to isHProp(A) \to A \\ &choice\ |a|\ \_ \equiv a \end{aligned}$$

Problem 1: the proof of $\|A\|$ might contain axioms, so we would lose canonicity

Observational choice

What if we give up proof-irrelevance? Then we could imagine

$$\begin{aligned} &choice : \|A\| \to isHProp(A) \to A \\ &choice\ |a|\ \_ \ \equiv\ a \end{aligned}$$

Problem 1: the proof of $\|A\|$ might contain axioms, so we would lose canonicity

Problem 2: in an inconsistent context, we can use $\|Type\|$ as a universe which lives inside of Prop, which allows us to build non-terminating terms.

So, what are our options?

Maybe we can build a version of OTT that is

- Proof-relevant (no definitional UIP)
- Axiom-free
- Careful with the interaction of choice and impredicativity

Losing definitional UIP is a bit disappointing! But that just might be the price we have to pay in exchange for bits of choice.

3. Toward a Proof-Relevant OTT

Relevant Observations

The definition of a relevant observational equality does not change much.

$$\frac{A : Type_\ell \qquad B : Type_\ell \qquad a : A \qquad b : B}{a\ {}_A{\sim}_B\, b : Prop}$$

Relevant Observations

The definition of a relevant observational equality does not change much.

$$\frac{A : Type_\ell \qquad B : Type_\ell \qquad a : A \qquad b : B}{a \;{}_A{\sim}_B\, b : Prop}$$

It lands in Prop (not SProp), and computes on type constructors:

- On Π-types, equality is defined pointwise
- On Σ-types, it is the (dependent) equality of both projections
- On Type, it is the equality of codes
- On Prop, it is the logical equivalence
- On incompatible type formers, it is False

Groupoid Laws

However, we cannot get away with postulating groupoid laws as axioms anymore, lest we get stuck terms in Prop.

Groupoid Laws

However, we cannot get away with postulating groupoid laws as axioms anymore, lest we get stuck terms in Prop.

We need:

- function congruence

Groupoid Laws

However, we cannot get away with postulating groupoid laws as axioms anymore, lest we get stuck terms in Prop.

We need:

- function congruence

$$\frac{e : x \;{}_{A}{\sim}_{A}\; y \qquad f : \Pi\, A\, B}{\mathit{cong}\; f\; e : (f\; x) \;{}_{B[x]}{\sim}_{B[y]}\; (f\; y)}$$

But this is in fact equivalent to a proof of $f \sim f$
Thus function congruence is subsumed by reflexivity.

Groupoid Laws

However, we cannot get away with postulating groupoid laws as axioms anymore, lest we get stuck terms in Prop.

We need:

- function congruence
- transitivity

Groupoid Laws

However, we cannot get away with postulating groupoid laws as axioms anymore, lest we get stuck terms in Prop.

We need:

- function congruence
- transitivity

$$\frac{e_1 : x \ {}_A{\sim}_A \ y \qquad e_2 : y \ {}_A{\sim}_A \ z}{e_1 \cdot e_2 : x \ {}_A{\sim}_A \ z}$$

Transitivity can be obtained from congruence of $\lambda(y : A).x \ {}_A{\sim}_A \ y$ and cast.

Groupoid Laws

However, we cannot get away with postulating groupoid laws as axioms anymore, lest we get stuck terms in Prop.

We need:

- function congruence
- transitivity
- symmetry

Groupoid Laws

However, we cannot get away with postulating groupoid laws as axioms anymore, lest we get stuck terms in Prop.

We need:

- function congruence
- transitivity
- symmetry

$$\frac{e : x \ {}_A{\sim}_A \ y}{e^{-1} : y \ {}_A{\sim}_A \ x}$$

Symmetry can be added with "backward cast", which is no more difficult than cast

Groupoid Laws

However, we cannot get away with postulating groupoid laws as axioms anymore, lest we get stuck terms in Prop.

We need:

- function congruence
- transitivity
- symmetry
- reflexivity

Groupoid Laws

However, we cannot get away with postulating groupoid laws as axioms anymore, lest we get stuck terms in Prop.

We need:

- function congruence
- transitivity
- symmetry
- reflexivity

In the end, reflexivity is the main obstacle (assuming we can do cast)

For this one, we are going to explore an idea from Higher Observational Type Theory [Altenkirch et al 2023] (itself echoing ideas from Internal Parametricity [Bernardy et al 2012])

Reflexivity

The definition of the observational equality coincides with the binary parametricity translation for an inductive-recursive universe.

Reflexivity

The definition of the observational equality coincides with the binary parametricity translation for an inductive-recursive universe.

Given a term in context t

$$\Gamma \vdash t : C$$

The binary parametricity translation produces a new term

$$[\![\Gamma]\!] \vdash [t]_\varepsilon : [\![C]\!]_\varepsilon \, [t]_0 \, [t]_1$$

Where $[\![\Gamma]\!]$ duplicates all the variables of Γ:

$$\begin{array}{lcl} [\![\cdot]\!] & := & \cdot \\ [\![\Gamma, x : A]\!] & := & [\![\Gamma]\!], x_0 : [\![A]\!]_0, x_1 : [\![A]\!]_1, x_0 : [\![A]\!]_\varepsilon \, x_0 \, x_1 \end{array}$$

Reflexivity

The definition of the observational equality coincides with the binary parametricity translation for an inductive-recursive universe.

Reflexivity

The definition of the observational equality coincides with the binary parametricity translation for an inductive-recursive universe.

We start with

$$[\![Prop]\!]_\varepsilon\, t\, u \;:=\; t \leftrightarrow u$$

and we unroll the usual translation from there:

$$\begin{aligned}
[\![\Pi A B]\!]_\varepsilon\, f\, g \quad &:= \quad \Pi\,(a_0 : [\![A]\!]_0)\,(a_1 : [\![A]\!]_1)\,(a_\varepsilon : [\![A]\!]_\varepsilon a_0 a_1)\,.\\
&\qquad\quad [\![B]\!]_\varepsilon\,(f\,a_0)\,(g\,a_1)\\
[\![\Sigma A B]\!]_\varepsilon\, t\, u \quad &:= \quad \Sigma\,(a_\varepsilon : [\![A]\!]_\varepsilon\,(fst\,t)\,(fst\,u))\,.\\
&\qquad\quad [\![B]\!]_\varepsilon\{a_0 := fst\,t\,;\ a_1 := fst\,u\}\,(snd\,t)\,(snd\,u)
\end{aligned}$$

Observe that $[\![A]\!]_\varepsilon\, t\, u$ coincides with $t\ {}_{A_0 \sim A_1}\ u$

Reflexivity

Of course, the parametricity translation also applies to terms.

$$[\![\Gamma]\!] \vdash [t]_\varepsilon : [\![C]\!]_\varepsilon \, [t]_0 \, [t]_1$$

→ $[t]_\varepsilon$ plays the role of the (heterogeneous) reflexivity proof for t.

Of course, $[t]_\varepsilon$ is defined in the duplicated context $[\![\Gamma]\!]$. In order to get a proper homogeneous reflexivity proof, we must substitute $[t]_\varepsilon$ with reflexive terms.

Reflexivity

Of course, the parametricity translation also applies to terms.

$$[\![\Gamma]\!] \vdash [t]_\varepsilon : [\![C]\!]_\varepsilon \, [t]_0 \, [t]_1$$

→ $[t]_\varepsilon$ plays the role of the (heterogeneous) reflexivity proof for t.

Of course, $[t]_\varepsilon$ is defined in the duplicated context $[\![\Gamma]\!]$. In order to get a proper homogeneous reflexivity proof, we must substitute $[t]_\varepsilon$ with reflexive terms.

$$x : A \vdash t : B$$

Reflexivity

Of course, the parametricity translation also applies to terms.

$$\llbracket \Gamma \rrbracket \vdash [t]_\varepsilon : \llbracket C \rrbracket_\varepsilon \, [t]_0 \, [t]_1$$

→ $[t]_\varepsilon$ plays the role of the (heterogeneous) reflexivity proof for t.

Of course, $[t]_\varepsilon$ is defined in the duplicated context $\llbracket \Gamma \rrbracket$. In order to get a proper homogeneous reflexivity proof, we must substitute $[t]_\varepsilon$ with reflexive terms.

$$x : A \vdash t : B$$

$$x : A, x_e : x \ {}_{A}{\sim}_{A} \ x \vdash [t]_\varepsilon\{x_0 := x \,;\, x_1 := x \,;\, x_\varepsilon := x_e\} : t \ {}_{B}{\sim}_{B} \ t$$

Reflexivity

Of course, the parametricity translation also applies to terms.

$$[\![\Gamma]\!] \vdash [t]_\varepsilon : [\![C]\!]_\varepsilon \, [t]_0 \, [t]_1$$

→ $[t]_\varepsilon$ plays the role of the (heterogeneous) reflexivity proof for t.

Of course, $[t]_\varepsilon$ is defined in the duplicated context $[\![\Gamma]\!]$. In order to get a proper homogeneous reflexivity proof, we must substitute $[t]_\varepsilon$ with reflexive terms.

$$x : A \vdash t : B$$

$$x : A, x_e : x \;{}_A\!\sim_A x \vdash [t]_\varepsilon\{x_0 := x \,;\; x_1 := x \,;\; x_\varepsilon := x_e\} : t \;{}_B\!\sim_B t$$

Thus, by packing terms with their reflexivity proofs, we can build a model of type theory with a reflexive observational equality.

Type Casting

Now that we have the groupoid laws, it remains to define the cast operator
We define it mutually with a casteq operator (since the computation rule castrefl is not available anymore)

$$\frac{A : Type \qquad B : Type \qquad e : A \sim B \qquad t : A}{cast(A, B, e, t) : B}$$

$$\frac{A : Type \qquad B : Type \qquad e : A \sim B \qquad t : A}{casteq(A, B, e, t) : t \;{}_{A \sim B}\; cast(A, B, e, t)}$$

Their definition is by induction on the types, following McBride et al.

Axiom-free OTT

This is sufficient to define a version of OTT without axioms in the propositional layer.
It comes at a price: definitional UIP and computation of cast on refl.

Now, it seems to me that there are two directions to extend this base with choice principles.

Impredicativity + Acc elimination

It seems easy to add an accessibility predicate in Prop with large elimination.

Is the resulting theory well-behaved? Mixing impredicativity with primitives that compute by induction on the (predicative) universes is scary!

$\longrightarrow$ realisability-like semantics?

Unique choice

We want to add an operator

$$\mathit{unique} : (A : \mathit{Type}_\ell) \to \mathit{isHProp}\, A \to \|A\| \to A$$
$$\mathit{unique}\, A\, H_A\, |a| = a$$

Unique choice

We want to add an operator

$$unique : (A : Type_\ell) \to isHProp\, A \to \|A\| \to A$$
$$unique\, A\, H_A\, |a| = a$$

Problem 1: if the theory is impredicative, this computation rule causes non-termination on open terms

Unique choice

We want to add an operator

$$unique : (A : Type_\ell) \to isHProp\ A \to \|A\| \to A$$
$$unique\ A\ H_A\ |a| = a$$

Problem 1: if the theory is impredicative, this computation rule causes non-termination on open terms

Problem 2: $refl\ (unique\ A\ H_A\ x)$ is most naturally defined by using H_A
But $refl\ (unique\ A\ H_A\ |a|)$ should be convertible to $refl\ a$!

Thank you!