OTT

Observational Equality

meets

The Calculus of Inductive Constructions

CIC

Loïc Pujet, Nicolas Tabareau

10 april 2024

The Calculus of Inductive Constructions

Both Coq and Lean are based on the CIC

- Dependent type theory
- with a infinite universe hierarchy,
- an impredicative sort for propositions
- and a powerful scheme for inductive definitions

The Calculus of Inductive Constructions

Both Coq and Lean are based on the CIC

- Dependent type theory
- with a infinite universe hierarchy,
- an impredicative sort for propositions
- and a powerful scheme for inductive definitions

But difficulties with function extensionality and quotient types

Observational Equality

In observational type theories[1] the inductive equality is replaced with the observational equality:

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : A}{\Gamma \vdash t \sim_A u : SProp}$$

[1] Altenkirch, McBride, Swierstra '07 – Observational Equality, Now!

Observational Equality

In observational type theories[1] the inductive equality is replaced with the observational equality:

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : A}{\Gamma \vdash t \sim_A u : SProp}$$

Observational equality is eliminated via typecasting:

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

which computes by case analysis on A and B.

[1] Altenkirch, McBride, Swierstra '07 – Observational Equality, Now!

Observational Equality

In ordinary dependent type theory each type former comes with

- a type formation rule
- introduction rules
- elimination rules
- computation rules

Observational Equality

In ordinary dependent type theory each type former comes with

- a type formation rule
- introduction rules
- elimination rules
- computation rules

In observational type theory, every type former is also equipped with

- a definition for the equality between inhabitants
- a definition for the equality between two instances of the type
- computation rules for type-casting

Observational Equality

Let us look at the example of (nondependent) function types:

Observational Equality

Let us look at the example of (nondependent) function types:

- A definition for the equality between inhabitants

$$f \sim_{A \to B} g \quad \leftrightarrow \quad \Pi(x : A) \,.\, f\ x \sim_B g\ x$$

Observational Equality

Let us look at the example of (nondependent) function types:

- A definition for the equality between inhabitants

$$f \sim_{A \to B} g \quad \leftrightarrow \quad \Pi(x : A) \,.\, f\, x \sim_B g\, x$$

- A definition for the equality between two instances of the type former

$$(A \to B) \sim_{Type} (C \to D) \quad \leftrightarrow \quad (C \sim_{Type} A) \wedge (B \sim_{Type} D)$$

Observational Equality

Let us look at the example of (nondependent) function types:

- A definition for the equality between inhabitants

$$f \sim_{A \to B} g \quad \leftrightarrow \quad \Pi(x : A) \,.\, f\, x \sim_B g\, x$$

- A definition for the equality between two instances of the type former

$$(A \to B) \sim_{Type} (C \to D) \quad \leftrightarrow \quad (C \sim_{Type} A) \wedge (B \sim_{Type} D)$$

- A computation rule for type-casting

$$\begin{aligned} & cast(A \to B, C \to D, e, f) \\ & \quad \equiv \quad \lambda\, (x : C) \,.\, cast(B, D, e_2, f\, cast(C, A, e_1, x)) \end{aligned}$$

Observational Inductives?

Observational type theory is compatible with

Observational Inductives?

Observational type theory is compatible with

- Dependent products

Observational Inductives?

Observational type theory is compatible with

- Dependent products
- Universes

Observational Inductives?

Observational type theory is compatible with

- Dependent products
- Universes
- Impredicative strict Prop

Observational Inductives?

Observational type theory is compatible with

- Dependent products
- Universes
- Impredicative strict Prop
- Σ-types, natural numbers

Observational Inductives?

Observational type theory is compatible with

- Dependent products
- Universes
- Impredicative strict Prop
- Σ-types, natural numbers

How do we fit the general inductive definitions of CIC into this picture?

First Example: Lists

$$
\begin{array}{l}
\mathit{Inductive}\ \ \mathit{list}\ \ (A : \mathit{Type}_\ell) : \mathit{Type}_\ell := \\
\mid \mathit{nil} : \mathit{list}\, A \\
\mid \mathit{cons} : A \to \mathit{list}\, A \to \mathit{list}\, A
\end{array}
$$

First Example: Lists

$$\begin{aligned}&\mathit{Inductive}\ \mathit{list}\ (A : \mathit{Type}_\ell) : \mathit{Type}_\ell :=\\ &\mid \mathit{nil} : \mathit{list}\, A\\ &\mid \mathit{cons} : A \to \mathit{list}\, A \to \mathit{list}\, A\end{aligned}$$

- When should two inhabitants of $\mathit{list}\, A$ be equal?

First Example: Lists

$$\textit{Inductive list } (A : \textit{Type}_\ell) : \textit{Type}_\ell :=$$
$$\mid \textit{nil} : \textit{list } A$$
$$\mid \textit{cons} : A \to \textit{list } A \to \textit{list } A$$

- When should two inhabitants of $\textit{list } A$ be equal?

  The J eliminator already gives the correct answer!

First Example: Lists

$$\begin{aligned}&\mathit{Inductive}\ \ \mathit{list}\ \ (A : \mathit{Type}_\ell) : \mathit{Type}_\ell :=\\ &\mid \mathit{nil} : \mathit{list}\, A\\ &\mid \mathit{cons} : A \to \mathit{list}\, A \to \mathit{list}\, A\end{aligned}$$

- When should two inhabitants of $\mathit{list}\ A$ be equal?

  The J eliminator already gives the correct answer!
- When should $\mathit{list}\ A$ and $\mathit{list}\ B$ be equal types?

First Example: Lists

$$\textit{Inductive}\ \ list\ \ (A : Type_\ell) : Type_\ell :=$$
$$|\ nil : list\ A$$
$$|\ cons : A \to list\ A \to list\ A$$

- When should two inhabitants of $list\ A$ be equal?

  The J eliminator already gives the correct answer!

- When should $list\ A$ and $list\ B$ be equal types?

  $list\text{-}eq\ \ :\ \ list\ A \sim list\ B\ \ \to\ \ A \sim B.$

First Example: Lists

$$
\begin{array}{l}
\mathit{Inductive}\ \ \mathit{list}\ \ (A : \mathit{Type}_\ell) : \mathit{Type}_\ell :=\\
\mid \mathit{nil} : \mathit{list}\, A\\
\mid \mathit{cons} : A \to \mathit{list}\, A \to \mathit{list}\, A
\end{array}
$$

- When should two inhabitants of $\mathit{list}\ A$ be equal?

  The J eliminator already gives the correct answer!

- When should $\mathit{list}\ A$ and $\mathit{list}\ B$ be equal types?

  $\mathit{list\text{-}eq} \;:\; \mathit{list}\ A \sim \mathit{list}\ B \;\to\; A \sim B.$

- How does type-casting compute?

First Example: Lists

$$\begin{aligned}&\mathit{Inductive}\ \ \mathit{list}\ \ (A : \mathit{Type}_\ell) : \mathit{Type}_\ell := \\ &\mid \mathit{nil} : \mathit{list}\,A \\ &\mid \mathit{cons} : A \to \mathit{list}\,A \to \mathit{list}\,A\end{aligned}$$

- When should two inhabitants of $\mathit{list}\,A$ be equal?

  The J eliminator already gives the correct answer!

- When should $\mathit{list}\,A$ and $\mathit{list}\,B$ be equal types?

  $\mathit{list\text{-}eq}\ :\ \mathit{list}\,A \sim \mathit{list}\,B \ \to\ A \sim B.$

- How does type-casting compute?

$$\begin{aligned}&\mathit{cast}(\mathit{list}\,A, \mathit{list}\,B, e, \mathit{nil}) &&\equiv\ \mathit{nil} \\ &\mathit{cast}(\mathit{list}\,A, \mathit{list}\,B, e, \mathit{cons}\ a\ l) &&\equiv \\ &\qquad \mathit{cons}\ \ \mathit{cast}(A, B, \mathit{list\text{-}eq}\ e, a)\ \ \mathit{cast}(\mathit{list}\,A, \mathit{list}\,B, e, l)\end{aligned}$$

Second Example: Inductive Equality

$$\mathit{Inductive}\ \ eq\ \ (A : Type_{\ell})(x : A) : A \to Type_0 :=$$
$$|\ eq\_refl : eq\ A\ x\ x$$

Second Example: Inductive Equality

$$\mathit{Inductive}\ \ eq\ \ (A : \mathit{Type}_\ell)(x : A) : A \to \mathit{Type}_0 :=$$
$$|\ eq\_refl : eq\ A\ x\ x$$

- When should two inhabitants of $eq\ A\ x\ y$ be equal?

Second Example: Inductive Equality

$$\textit{Inductive } eq\ (A : Type_\ell)(x : A) : A \to Type_0 :=$$
$$\mid eq\_refl : eq\ A\ x\ x$$

- When should two inhabitants of $eq\ A\ x\ y$ be equal?

  The J eliminator does not seem to be sufficient to prove UIP ☹

Second Example: Inductive Equality

$$\textit{Inductive}\ \ eq\ \ (A : Type_\ell)(x : A) : A \to Type_0 :=$$
$$\mid eq\_refl : eq\ A\ x\ x$$

- When should two inhabitants of $eq\ A\ x\ y$ be equal?

  The J eliminator does not seem to be sufficient to prove UIP ☹
- When should $eq\ A\ x\ y$ and $eq\ A'\ x'\ y'$ be equal types?

Second Example: Inductive Equality

$$\mathit{Inductive}\ eq\ (A : \mathit{Type}_\ell)(x : A) : A \to \mathit{Type}_0 :=$$
$$|\ eq\_refl : eq\ A\ x\ x$$

- When should two inhabitants of $eq\ A\ x\ y$ be equal?

  The J eliminator does not seem to be sufficient to prove UIP ☹
- When should $eq\ A\ x\ y$ and $eq\ A'\ x'\ y'$ be equal types?

  Equality of the parameters and indices?

Second Example: Inductive Equality

$$\mathit{Inductive}\ eq\ (A : \mathit{Type}_\ell)(x : A) : A \to \mathit{Type}_0 :=$$
$$\mid eq\_refl : eq\ A\ x\ x$$

- When should two inhabitants of $eq\ A\ x\ y$ be equal?

  The J eliminator does not seem to be sufficient to prove UIP ☹
- When should $eq\ A\ x\ y$ and $eq\ A'\ x'\ y'$ be equal types?

  Equality of the parameters and indices?
  $\to$ eq becomes an injective function from Type_ℓ to Type_0

Second Example: Inductive Equality

$$\mathit{Inductive}\ \ eq\ \ (A : \mathit{Type}_\ell)(x : A) : A \to \mathit{Type}_0 :=$$
$$|\ eq\_refl : eq\ A\ x\ x$$

- When should two inhabitants of $eq\ A\ x\ y$ be equal?

  The J eliminator does not seem to be sufficient to prove UIP ☹
- When should $eq\ A\ x\ y$ and $eq\ A'\ x'\ y'$ be equal types?

  Equality of the parameters and indices?
  $\to$ eq becomes an injective function from Type_ℓ to Type_0
  $\to$ universe inconsistencies ☹

Second Example: Inductive Equality

$\mathit{Inductive}\ eq\ (A : \mathit{Type}_\ell)(x : A) : A \to \mathit{Type}_0 :=$
$\mid eq\_refl : eq\ A\ x\ x$

- When should two inhabitants of $eq\ A\ x\ y$ be equal?

  The J eliminator does not seem to be sufficient to prove UIP ☹
- When should $eq\ A\ x\ y$ and $eq\ A'\ x'\ y'$ be equal types?

  Equality of the parameters and indices?
  $\rightarrow$ eq becomes an injective function from Type_ℓ to Type_0
  $\rightarrow$ universe inconsistencies ☹
- How does type-casting compute?

Second Example: Inductive Equality

$$\mathit{Inductive}\ \ eq\ \ (A : \mathit{Type}_\ell)(x : A) : A \to \mathit{Type}_0 :=$$
$$\mid eq\_refl : eq\ A\ x\ x$$

- When should two inhabitants of $eq\ A\ x\ y$ be equal?

  The J eliminator does not seem to be sufficient to prove UIP ☹

- When should $eq\ A\ x\ y$ and $eq\ A'\ x'\ y'$ be equal types?

  Equality of the parameters and indices?
  → eq becomes an injective function from Type_ℓ to Type_0
  → universe inconsistencies ☹

- How does type-casting compute?

$$cast(eq\ A\ x\ x,\ eq\ A'\ x'\ y',\ e,\ eq\_refl) \equiv \cancel{eq\_refl}$$

Second Example: Inductive Equality

$$\textit{Inductive } eq\ (A : Type_\ell)(x : A) : A \to Type_0 :=$$
$$\mid eq\_refl : eq\ A\ x\ x$$

- When should two inhabitants of $eq\ A\ x\ y$ be equal?

  The J eliminator does not seem to be sufficient to prove UIP ☹

- When should $eq\ A\ x\ y$ and $eq\ A'\ x'\ y'$ be equal types?

  Equality of the parameters and indices?
  $\to$ eq becomes an injective function from $Type_\ell$ to $Type_0$
  $\to$ universe inconsistencies ☹

- How does type-casting compute?

$$cast(eq\ A\ x\ x, eq\ A'\ x'\ y', e, eq\_refl) \equiv \cancel{eq\_refl}$$

  Does not typecheck ☹

Observational Inductives?

Not so simple!

Constructor arguments, not parameters!

The universe inconsistency shows up because the size of an inductive is determined by the types of its constructor arguments, not parameters or indices.

$$\mathit{Inductive}\ \ \mathit{Small}\ \ (A : \mathit{Type}_\ell) : \mathit{Type}_0 :=$$
$$\mid \mathit{small} : \mathbb{N} \to \mathit{Small}\ A$$

$$\mathit{Inductive}\ \ \mathit{Large}\ \ (A : \mathit{Type}_\ell) : \mathit{Type}_\ell :=$$
$$\mid \mathit{large} : A \to \mathit{Large}\ A$$

Constructor arguments, not parameters!

The universe inconsistency shows up because the size of an inductive is determined by the types of its constructor arguments, not parameters or indices.

$$\mathit{Inductive}\ \ \mathit{Small}\ \ (A : \mathit{Type}_{\ell}) : \mathit{Type}_0 :=$$
$$\mid \mathit{small} : \mathbb{N} \to \mathit{Small}\ A$$

the type of
the argument
is large

the type of
the argument
is small

$$\mathit{Inductive}\ \ \mathit{Large}\ \ (A : \mathit{Type}_{\ell}) : \mathit{Type}_{\ell} :=$$
$$\mid \mathit{large} : A \to \mathit{Large}\ A$$

Constructor arguments, not parameters!

Lazy way out: bump up the universe levels of the inductives according to their parameters and indices.

Constructor arguments, not parameters!

Lazy way out: bump up the universe levels of the inductives according to their parameters and indices.

- reasonable for indices (cf HoTT)

Constructor arguments, not parameters!

Lazy way out: bump up the universe levels of the inductives according to their parameters and indices.

- reasonable for indices (cf HoTT)
- unacceptable for parameters!

Constructor arguments, not parameters!

Better way out: equality of inductive types should imply the equality of the types of the constructor arguments.

Constructor arguments, not parameters!

Better way out: equality of inductive types should imply the equality of the types of the constructor arguments.

$$\mathit{Inductive}\ \ \mathit{Small}\ \ (A : \mathit{Type}_\ell) : \mathit{Type}_0 :=$$
$$\mid \mathit{small} : \mathbb{N} \to \mathit{Small}\ A$$

$$\mathit{eq\text{-}Small} : \mathit{Small}\ A \sim \mathit{Small}\ B \quad \to \quad \mathbb{N} \sim \mathbb{N}$$

Constructor arguments, not parameters!

Better way out: equality of inductive types should imply the equality of the types of the constructor arguments.

$$\textit{Inductive}\ \ Small\ \ (A : Type_\ell) : Type_0 :=$$
$$\mid small : \mathbb{N} \to Small\ A$$

$$eq\text{-}Small : Small\ A \sim Small\ B \quad \to \quad \mathbb{N} \sim \mathbb{N}$$

$$\textit{Inductive}\ \ Large\ \ (A : Type_\ell) : Type_\ell :=$$
$$\mid large : A \to Large\ A$$

$$eq\text{-}Large : Large\ A \sim Large\ B \quad \to \quad A \sim B$$

Constructor arguments, not parameters!

Better way out: equality of inductive types should imply the equality of the types of the constructor arguments.

$$\mathit{Inductive}\ \ \mathit{Small}\ \ (A : \mathit{Type}_\ell) : \mathit{Type}_0 :=$$
$$\mid \mathit{small} : \mathbb{N} \to \mathit{Small}\ A$$

$$\mathit{eq\text{-}Small} : \mathit{Small}\ A \sim \mathit{Small}\ B \quad \to \quad \mathbb{N} \sim \mathbb{N}$$

$$\mathit{Inductive}\ \ \mathit{Large}\ \ (A : \mathit{Type}_\ell) : \mathit{Type}_\ell :=$$
$$\mid \mathit{large} : A \to \mathit{Large}\ A$$

$$\mathit{eq\text{-}Large} : \mathit{Large}\ A \sim \mathit{Large}\ B \quad \to \quad A \sim B$$

$$\mathit{cast}(\mathit{Small}\ A, \mathit{Small}\ B, e, \mathit{small}\ n) \equiv \mathit{small}\ \mathit{cast}(\mathbb{N}, \mathbb{N}, \mathit{eq\text{-}Small}\ e, n)$$
$$\mathit{cast}(\mathit{Large}\ A, \mathit{Large}\ B, e, \mathit{large}\ x) \equiv \mathit{large}\ \mathit{cast}(A, B, \mathit{eq\text{-}Large}\ e, x)$$

Observational Inductives?

With this technique, we can smoothly handle all inductive definitions without indices

$$\mathit{Inductive}\ \ eq\ \ \overbrace{(A : Type_\ell)(x : A)}^{\text{Parameters}} : \overbrace{\Pi(x : A)}^{\text{Indices}}.Type_0 :=$$
$$|\ eq\_refl : eq\ A\ x\ x$$

Observational Inductives?

With this technique, we can smoothly handle all inductive definitions without indices

$$\textit{Inductive}\ eq\ \underbrace{(A : \textit{Type}_\ell)(x : A)}_{\text{Parameters}} : \underbrace{\Pi (x : A)}_{\text{Indices}} . \textit{Type}_0 :=$$
$$\mid eq\_refl : eq\ A\ x\ x$$

Treating indices will require a few more tricks.

No Canonicity for Indices

Remember our failed attempt at a computation rule

$$\mathit{Inductive}\ \ eq\ \ (A : \mathit{Type}_\ell)(x : A) : A \to \mathit{Type}_0 :=$$
$$\mid eq\_refl : eq\ A\ x\ x$$

$$cast(eq\ A\ x\ x,\ eq\ A'\ x'\ y',\ e,\ eq\_refl) \ \equiv\ \cancel{eq\_refl}$$

No Canonicity for Indices

Remember our failed attempt at a computation rule

$$\mathit{Inductive}\ \ eq\ \ (A : \mathit{Type}_\ell)(x : A) : A \to \mathit{Type}_0 :=$$
$$|\ eq\_refl : eq\ A\ x\ x$$

$$cast(eq\ A\ x\ x,\ eq\ A'\ x'\ y',\ e,\ eq\_refl) \ \equiv\ \cancel{eq\_refl}$$

We cannot simplify casts on indices in general...

No Canonicity for Indices

Remember our failed attempt at a computation rule

$$\mathit{Inductive}\ \ eq\ \ (A : \mathit{Type}_\ell)(x : A) : A \to \mathit{Type}_0 :=$$
$$|\ eq\_refl : eq\ A\ x\ x$$

$$cast(eq\ A\ x\ x,\ eq\ A'\ x'\ y',\ e,\ eq\_refl)\ \equiv\ \cancel{eq\_refl}$$

We cannot simplify casts on indices in general...

...but we can encode them away with observational equality

"You can pick any colour, as long as it is black"

Henry Ford's trick[2] can be used to encode indices with equalities on parameters:

[2] Altenkirch, McBride '06 – Towards Observational Type Theory

"You can pick any colour, as long as it is black"

Henry Ford's trick[2] can be used to encode indices with equalities on parameters:

$$
\begin{array}{l}
\mathit{Inductive}\ \ \mathit{vector}\ \ (A : \mathit{Type}_\ell) : \mathbb{N} \to \mathit{Type}_\ell :=\\
\mid \mathit{vnil} : \mathit{vector}\ A\ 0\\
\mid \mathit{vcons} : \Pi\,(m : \mathbb{N}).\, A \to \mathit{vector}\ A\ m \to \mathit{vector}\ A\ (S\ m)
\end{array}
$$

[2] Altenkirch, McBride '06 – Towards Observational Type Theory

"You can pick any colour, as long as it is black"

Henry Ford's trick[2] can be used to encode indices with equalities on parameters:

$$
\begin{array}{l}
\mathit{Inductive}\ \ \mathit{vector}\ \ (A : \mathit{Type}_\ell) : \mathbb{N} \to \mathit{Type}_\ell :=\\
\mid \mathit{vnil} : \mathit{vector}\ A\ 0\\
\mid \mathit{vcons} : \Pi\,(m : \mathbb{N}).\ A \to \mathit{vector}\ A\ m \to \mathit{vector}\ A\ (S\ m)
\end{array}
$$

becomes

$$
\begin{array}{l}
\mathit{Inductive}\ \ \mathit{vector}_F\ \ (A : \mathit{Type}_\ell)(n : \mathbb{N}) : \mathit{Type}_\ell :=\\
\mid \mathit{vnil}_F : (n \sim 0) \to \mathit{vector}_F\ A\ n\\
\mid \mathit{vcons}_F : \Pi\,(m : \mathbb{N}).\ A \to \mathit{vector}_F\ A\ m \to (n \sim S\ m) \to \mathit{vector}_F\ A\ n
\end{array}
$$

[2] Altenkirch, McBride '06 – Towards Observational Type Theory

"You can pick any colour, as long as it is black"

Henry Ford's trick[2] can be used to encode indices with equalities on parameters:

$$\begin{array}{l} \mathit{Inductive}\ \ \mathit{vector}\ \ (A : \mathit{Type}_\ell) : \mathbb{N} \to \mathit{Type}_\ell := \\ \mid \mathit{vnil} : \mathit{vector}\ A\ 0 \\ \mid \mathit{vcons} : \Pi\,(m : \mathbb{N}).\ A \to \mathit{vector}\ A\ m \to \mathit{vector}\ A\ (S\ m) \end{array}$$

becomes

$$\begin{array}{l} \mathit{Inductive}\ \ \mathit{vector}_F\ \ (A : \mathit{Type}_\ell)(n : \mathbb{N}) : \mathit{Type}_\ell := \\ \mid \mathit{vnil}_F : (n \sim 0) \to \mathit{vector}_F\ A\ n \\ \mid \mathit{vcons}_F : \Pi\,(m : \mathbb{N}).\ A \to \mathit{vector}_F\ A\ m \to (n \sim S\ m) \to \mathit{vector}_F\ A\ n \end{array}$$

and now we can used our recipe for inductives without indices.

---

[2] Altenkirch, McBride '06 – Towards Observational Type Theory

"You can pick any colour, as long as it is black"

In the case of the inductive equality, Henry Ford's encoding produces:

$$\textit{Inductive}\ \ eq_F\ \ (A : \mathit{Type}_\ell)(x : A)(y : A) : \mathit{Type}_0 :=$$
$$\mid eq\_refl_F : x \sim_A y \to eq_F\ A\ x\ y$$

"You can pick any colour, as long as it is black"

In the case of the inductive equality, Henry Ford's encoding produces:

$$\mathit{Inductive}\ \ eq_F\ \ (A : Type_\ell)(x : A)(y : A) : Type_0 :=$$
$$\mid eq\_refl_F : x \sim_A y \rightarrow eq_F\ A\ x\ y$$

→ an inhabitant of the inductive equality packs a hidden proof of the observational equality!

"You can pick any colour, as long as it is black"

In the case of the inductive equality, Henry Ford's encoding produces:

$$\begin{array}{l} \textit{Inductive}\ \ eq_F\ \ (A : Type_\ell)(x : A)(y : A) : Type_0 := \\ \mid eq\_refl_F : x \sim_A y \to eq_F\ A\ x\ y \end{array}$$

→ an inhabitant of the inductive equality packs a hidden proof of the observational equality!

Our strategy: present eq to the user but elaborate everything to eq_F under the hood.

$$\begin{array}{rcl} eq & \rightsquigarrow & eq_F \\ eq\_refl & \rightsquigarrow & eq\_refl_F\ refl \\ & \ldots & \end{array}$$

"You can pick any colour, as long as it is black"

$$eq\_elim \quad \rightsquigarrow \quad \ldots$$

We can write a term with the expected type using $cast$ and eq_F_elim

"You can pick any colour, as long as it is black"

$$eq\_elim \quad \rightsquigarrow \quad \ldots$$

We can write a term with the expected type using $cast$ and eq_F_elim

However, the computation rule is not preserved: $cast$ only computes on closed types, while eq_elim can compute even when the return type is open.

"You can pick any colour, as long as it is black"

$$eq\_elim \quad \rightsquigarrow \quad \ldots$$

We can write a term with the expected type using $cast$ and eq_F_elim

However, the computation rule is not preserved: $cast$ only computes on closed types, while eq_elim can compute even when the return type is open.

The missing ingredient is the computation rule for cast on reflexivity:

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

The missing rule

Our goal: adding $cast(A, A, refl, t) \equiv t$ as a definitional equality

The missing rule

Our goal: adding $cast(A, A, refl, t) \equiv t$ as a definitional equality

Because of proof irrelevance, it should apply whenever the two endpoints of the cast are convertible:

$$\frac{\Gamma \vdash A \equiv B}{\Gamma \vdash cast(A, B, e, t) \equiv t \,:\, B\ x}$$

The missing rule

Our goal: adding $cast(A, A, refl, t) \equiv t$ as a definitional equality

Because of proof irrelevance, it should apply whenever the two endpoints of the cast are convertible:

$$\frac{\Gamma \vdash A \equiv B}{\Gamma \vdash cast(A, B, e, t) \equiv t \,:\, B \; x}$$

$\rightarrow$ nonlinear reduction rule which specifies reduction mutually with conversion checking

Is this déjà vu?

This idea is reminiscent of Lean's treatment of the J eliminator:

$$\frac{P\ a \equiv P\ b}{J(A, a, P, t, b, e) \equiv t}$$

Is this déjà vu?

This idea is reminiscent of Lean's treatment of the J eliminator:

$$\frac{P\ a \equiv P\ b}{J(A, a, P, t, b, e) \equiv t}$$

⚠ Abel, Coquand '19 – Failure of Normalization in Impredicative Type Theory with Proof-Irrelevant Propositional Equality

Is this déjà vu?

This idea is reminiscent of Lean's treatment of the J eliminator:

$$\frac{P\ a \equiv P\ b}{J(A, a, P, t, b, e) \equiv t}$$

⚠ Abel, Coquand '19 – Failure of Normalization in Impredicative Type Theory with Proof-Irrelevant Propositional Equality

This is not an undecidability result, though

> Does the addition of Werner's rule, while destroying proof normalization, preserve decidability of conversion and type checking? (Since proofs are irrelevant for equality, they need not be normalized during type checking.)

The conversion checking algorithm

Because cast reduces on type constructors, this rule only plays a role for relevant neutral terms.

The conversion checking algorithm

Because cast reduces on type constructors, this rule only plays a role for relevant neutral terms.

- we can implement it as a nonlinear reduction rule

The conversion checking algorithm

Because cast reduces on type constructors, this rule only plays a role for relevant neutral terms.

- we can implement it as a nonlinear reduction rule
- or we can offload it to the conversion checker for neutral terms.

The conversion checking algorithm

Because cast reduces on type constructors, this rule only plays a role for relevant neutral terms.

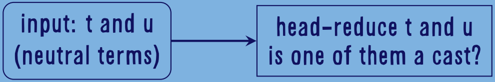
- we can implement it as a nonlinear reduction rule
- or we can offload it to the conversion checker for neutral terms.

input: t and u
(neutral terms)

The conversion checking algorithm

Because cast reduces on type constructors, this rule only plays a role for relevant neutral terms.

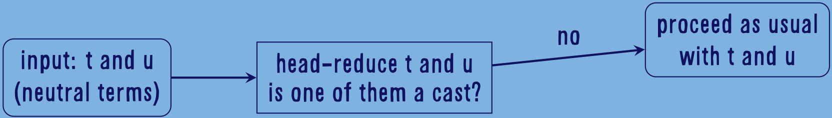
- we can implement it as a nonlinear reduction rule
- or we can offload it to the conversion checker for neutral terms.



The conversion checking algorithm

Because cast reduces on type constructors, this rule only plays a role for relevant neutral terms.

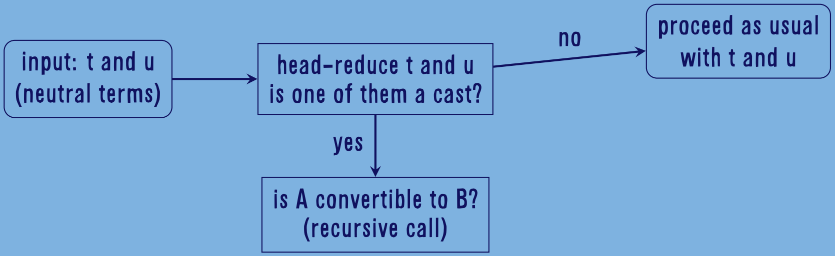
- we can implement it as a nonlinear reduction rule
- or we can offload it to the conversion checker for neutral terms.



The conversion checking algorithm

Because cast reduces on type constructors, this rule only plays a role for relevant neutral terms.

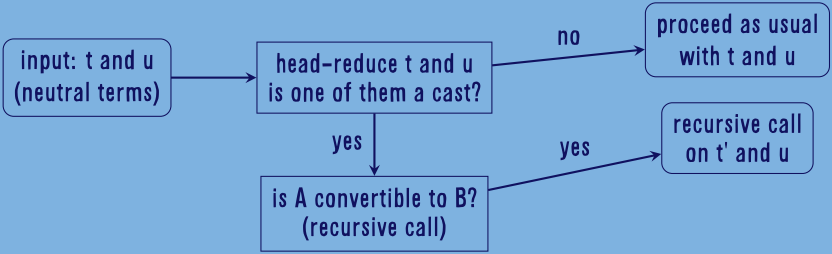
- we can implement it as a nonlinear reduction rule
- or we can offload it to the conversion checker for neutral terms.



The conversion checking algorithm

Because cast reduces on type constructors, this rule only plays a role for relevant neutral terms.

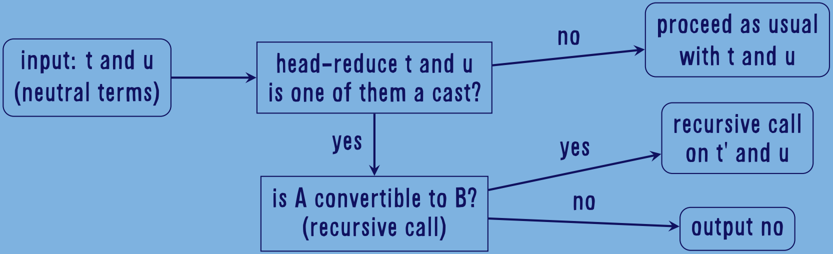
- we can implement it as a nonlinear reduction rule
- or we can offload it to the conversion checker for neutral terms.



The conversion checking algorithm

Because cast reduces on type constructors, this rule only plays a role for relevant neutral terms.

- we can implement it as a nonlinear reduction rule
- or we can offload it to the conversion checker for neutral terms.



Decidability proof

Using the second approach, we can add it on top of our logical relation model for $\mathsf{TT}^{obs}/\mathsf{CC}^{obs}$ [3]

[3] P, Tabareau '22 – Obserational Equality: Now for good

Decidability proof

Using the second approach, we can add it on top of our logical relation model for TT^{obs}/CC^{obs} [3]

→ Formal Agda proof of the decidability of conversion for our new rule

[3] P, Tabareau '22 – Obserational Equality: Now for good

Coming soon-ish

In your favourite rooster-themed proof assistant!

```
Set Observational Inductives.

(* Declaring an inductive automaticall adds equalities and rewrite rules for cast *)
Inductive list (A : Type) : Type :=
| nil : list A
| cons : forall (a : A) (l : list A), list A.

Parameter A B : Type.
Parameter e : list A ~ list B.
Parameter a : A.

Eval cbn in (cast (list A) (list B) e [ a ]).
(* [ cast A B (obseq_cons_0 A B e) a ] *)
```