Towards Quotient Inductive Types in Observational Type Theory

Thiago Felicissimo & Nicolas Tabareau

31st International Conference on Types for Proofs and Programs

11 June 2025

Gallinette, INRIA, Nantes

Quotient Inductive Types (QITs)

Inductive Types allow the declaration of generators:

$$\text{Inductive List } (A : \text{Type}) : \text{Type} :=$$

$$\mid [] : \mathsf{List}\ A$$

$$\mid \_ :: \_ \ (x : A)(m : \mathsf{List}\ A) : \mathsf{List}\ A$$

Quotient Inductive Types (QITs)

Quotient Inductive Types (QITs) allow the declaration of generators *and equations*:

$$
\begin{aligned}
&\mathrm{Inductive\ MSet}\ (A : \mathrm{Type}) : \mathrm{Type} :=\\
&\quad \mid [\,] : \mathrm{MSet}\ A\\
&\quad \mid \_ :: \_\ (x : A)(m : \mathrm{MSet}\ A) : \mathrm{MSet}\ A\\
&\quad \mid \mathrm{MSet}_{=}\ (x\ y : A)(m : \mathrm{MSet}\ A) : (x :: y :: m) = (y :: x :: m)
\end{aligned}
$$

Quotient Inductive Types (QITs)

Quotient Inductive Types (QITs) allow the declaration of generators *and equations*:

$$
\begin{array}{l}
\text{Inductive MSet } (A : \text{Type}) : \text{Type} := \\
\quad \mid [] : \text{MSet } A \\
\quad \mid \_ :: \_ \ (x : A)(m : \text{MSet } A) : \text{MSet } A \\
\quad \mid \text{MSet}_{=} \ (x\ y : A)(m : \text{MSet } A) : (x :: y :: m) = (y :: x :: m)
\end{array}
$$

Implicitly an hSet (in this talk, I only consider theories with UIP).

Quotient Inductive Types (QITs)

Quotient Inductive Types (QITs) allow the declaration of generators *and equations*:

$$\text{Inductive MSet } (A : \text{Type}) : \text{Type} :=$$
$$\mid [\,] : \text{MSet } A$$
$$\mid \_ :: \_ \ (x : A)(m : \text{MSet } A) : \text{MSet } A$$
$$\mid \text{MSet}_= \ (x\ y : A)(m : \text{MSet } A) : (x :: y :: m) = (y :: x :: m)$$

Implicitly an hSet (in this talk, I only consider theories with UIP).

Functions eliminating a QIT must respect equality:

$$\text{Fixpoint sum } (l : \text{MSet Nat}) : \text{Nat} :=$$
$$\text{match } l \text{ with}$$
$$\mid [\,] \longrightarrow 0 \qquad \mid x :: m \longrightarrow x + (\text{sum } m)$$
$$\mid \text{MSet}_= \ x\ y\ m \longrightarrow (\ldots) : (x + y + \text{sum } m) = (y + x + \text{sum } m)$$

Observational Type Theory (OTT) to the rescue

Problem In ITT (Coq, Agda, Lean), equality axioms of QIT can block computation:

$$(\mathrm{match}\ (\mathit{MSet}_{=}\ x\ y\ m)\ \mathrm{with}\ |\ \mathrm{refl} \to 0) : \mathrm{Nat}$$

Observational Type Theory (OTT) to the rescue

Problem In ITT (Coq, Agda, Lean), equality axioms of QIT can block computation:

$$(\text{match } (\mathit{MSet}_{=}\ x\ y\ m) \text{ with } \mid \text{refl} \to 0) : \text{Nat}$$

In Altenkirch & McBride's *Observational Type Theory (OTT)*, equality is instead eliminated using a *cast* operator:

$$\frac{A, B : \text{Type} \qquad p : A =_{\text{Type}} B \qquad a : A}{\text{cast}_p^{A \rightsquigarrow B}(a) : B}$$

Observational Type Theory (OTT) to the rescue

Problem In ITT (Coq, Agda, Lean), equality axioms of QIT can block computation:

$$(\mathsf{match}\ (\mathsf{MSet}_{=}\ x\ y\ m)\ \mathsf{with}\ |\ \mathsf{refl} \to 0) : \mathsf{Nat}$$

In Altenkirch & McBride's *Observational Type Theory (OTT)*, equality is instead eliminated using a *cast* operator:

$$\frac{A, B : \mathrm{Type} \qquad p : A =_{\mathrm{Type}} B \qquad a : A}{\mathrm{cast}_p^{A \rightsquigarrow B}(a) : B}$$

Crucial property of OTT Computation rules for cast *never look inside eq. proofs!*

$$\mathrm{cast}_p^{(A \times B) \rightsquigarrow (A' \times B')}\, t \longrightarrow \langle \mathrm{cast}_{p.1}^{A \rightsquigarrow A'}(\pi_1 t), \mathrm{cast}_{p.2}^{B \rightsquigarrow B'}(\pi_2 t) \rangle$$

Observational Type Theory (OTT) to the rescue

Problem In ITT (Coq, Agda, Lean), equality axioms of QIT can block computation:

$$(\mathsf{match}\ (\mathsf{MSet}_{=}\ x\ y\ m)\ \mathsf{with}\ |\ \mathsf{refl} \to 0) : \mathsf{Nat}$$

In Altenkirch & McBride's *Observational Type Theory (OTT)*, equality is instead eliminated using a *cast* operator:

$$\frac{A, B : \mathsf{Type} \qquad p : A =_{\mathsf{Type}} B \qquad a : A}{\mathsf{cast}_p^{A \rightsquigarrow B}(a) : B}$$

Crucial property of OTT Computation rules for cast *never look inside eq. proofs!*

$$\mathsf{cast}_p^{(A \times B) \rightsquigarrow (A' \times B')}\, t \longrightarrow \langle \mathsf{cast}_{p.1}^{A \rightsquigarrow A'}(\pi_1 t), \mathsf{cast}_{p.2}^{B \rightsquigarrow B'}(\pi_2 t) \rangle$$

Thus, OTT accommodates desirable equality axioms (funext, propext, Q types) *without blocking computation.*

This work

We report on a WIP metatheoretic justification of (non-indexed) QITs in OTT, with dependent eliminators that compute *definitionally*.

This work

We report on a WIP metatheoretic justification of (non-indexed) QITs in OTT, with dependent eliminators that compute *definitionally*.

The plan

? Study metatheory of OTT + QIT scheme.

This work

We report on a WIP metatheoretic justification of (non-indexed) QITs in OTT, with dependent eliminators that compute *definitionally*.

The plan

✗ Study metatheory of OTT + QIT scheme.

This work

We report on a WIP metatheoretic justification of (non-indexed) QITs in OTT, with dependent eliminators that compute *definitionally*.

The plan

- ✗ Study metatheory of OTT + QIT scheme.
- ? Construct QITs from inductive types and Q (quotient types).

This work

We report on a WIP metatheoretic justification of (non-indexed) QITs in OTT, with dependent eliminators that compute *definitionally*.

The plan

- ✗ Study metatheory of OTT + QIT scheme.
- ✗ Construct QITs from inductive types and Q (quotient types).

This work

We report on a WIP metatheoretic justification of (non-indexed) QITs in OTT, with dependent eliminators that compute *definitionally*.

The plan

- ✗ Study metatheory of OTT + QIT scheme.
- ✗ Construct QITs from inductive types and Q (quotient types).
- ? Extend OTT with Fiore et al's *QW Types*.
  Show that all QITs can be constructed from QW types in OTT.[1]

[1] Construction was claimed by Fiore et al. in ETT, but only argued informally.

This work

We report on a WIP metatheoretic justification of (non-indexed) QITs in OTT, with dependent eliminators that compute *definitionally*.

The plan

- ✗ Study metatheory of OTT + QIT scheme.
- ✗ Construct QITs from inductive types and Q (quotient types).
- ✓ Extend OTT with Fiore et al's *QW Types*.
  Show that all QITs can be constructed from QW types in OTT.[1]

[1] Construction was claimed by Fiore et al. in ETT, but only argued informally.

This work

We report on a WIP metatheoretic justification of (non-indexed) QITs in OTT, with dependent eliminators that compute *definitionally*.

The plan

- ✗ Study metatheory of OTT + QIT scheme.
- ✗ Construct QITs from inductive types and Q (quotient types).
- ✓ Extend OTT with Fiore et al's *QW Types*.
  Show that all QITs can be constructed from QW types in OTT.[1]

Justification for extending Observational Rocq with a primitive scheme for QITs.

[1] Construction was claimed by Fiore et al. in ETT, but only argued informally.

Constructing QITs from QW in OTT

$$\text{QIT scheme} \longrightarrow \text{QW}$$

Constructing QITs from QW in OTT

$$\text{QIT scheme} \longrightarrow \text{QW ?}$$

Fiore et al.'s QW Types

$$\mathrm{Sig} = \mathrm{record}\ \{\mathrm{Op} : \mathrm{Type};\ \ \mathrm{Ar} : \mathrm{Op} \longrightarrow \mathrm{Type}\}$$

Fiore et al.'s QW Types

$$\mathrm{Sig} = \mathrm{record}\ \{\mathrm{Op} : \mathrm{Type};\ \ \mathrm{Ar} : \mathrm{Op} \longrightarrow \mathrm{Type}\}$$

$$\begin{aligned}
&\mathrm{Inductive}\ \overline{\mathrm{QW}}\ (\Sigma : \mathrm{Sig})(\Gamma : \mathrm{Type}) : \mathrm{Type} := \\
&\quad \mid \mathrm{var}\ (x : \Gamma) : \overline{\mathrm{QW}}\ \Sigma\ \Gamma \\
&\quad \mid \mathrm{op}\ (c : \Sigma.\mathrm{Op})\ (f : \Sigma.\mathrm{Ar}\ c \longrightarrow \overline{\mathrm{QW}}\ \Sigma\ \Gamma) : \overline{\mathrm{QW}}\ \Sigma\ \Gamma
\end{aligned}$$

Fiore et al.'s QW Types

$$\mathsf{Sig} = \mathsf{record}\ \{\mathsf{Op} : \mathsf{Type};\ \ \mathsf{Ar} : \mathsf{Op} \longrightarrow \mathsf{Type}\}$$

$$\begin{aligned}
&\mathsf{Inductive}\ \overline{\mathsf{QW}}\ (\Sigma : \mathsf{Sig})(\Gamma : \mathsf{Type}) : \mathsf{Type} := \\
&\quad \mid \mathsf{var}\ (x : \Gamma) : \overline{\mathsf{QW}}\ \Sigma\ \Gamma \\
&\quad \mid \mathsf{op}\ (c : \Sigma.\mathsf{Op})\ (f : \Sigma.\mathsf{Ar}\ c \longrightarrow \overline{\mathsf{QW}}\ \Sigma\ \Gamma) : \overline{\mathsf{QW}}\ \Sigma\ \Gamma
\end{aligned}$$

$$\mathsf{EqTh}\ \Sigma = \mathsf{record}\ \{\mathsf{E} : \mathsf{Type};\ \ \mathsf{Ctx} : \mathsf{E} \longrightarrow \mathsf{Type};\ \ \mathsf{lhs}, \mathsf{rhs} : (e : \mathsf{E}) \longrightarrow \overline{\mathsf{QW}}\ \Sigma\ (\mathsf{Ctx}\ e)\}$$

Fiore et al.'s QW Types

$$\mathsf{Sig} = \mathsf{record}\ \{\mathsf{Op} : \mathsf{Type};\ \ \mathsf{Ar} : \mathsf{Op} \longrightarrow \mathsf{Type}\}$$

$$\begin{aligned}
&\mathsf{Inductive}\ \overline{\mathsf{QW}}\ (\Sigma : \mathsf{Sig})(\Gamma : \mathsf{Type}) : \mathsf{Type} := \\
&\quad |\ \mathsf{var}\ (x : \Gamma) : \overline{\mathsf{QW}}\ \Sigma\ \Gamma \\
&\quad |\ \mathsf{op}\ (c : \Sigma.\mathsf{Op})\ (f : \Sigma.\mathsf{Ar}\ c \longrightarrow \overline{\mathsf{QW}}\ \Sigma\ \Gamma) : \overline{\mathsf{QW}}\ \Sigma\ \Gamma
\end{aligned}$$

$$\mathsf{EqTh}\ \Sigma = \mathsf{record}\ \{\mathsf{E} : \mathsf{Type};\ \ \mathsf{Ctx} : \mathsf{E} \longrightarrow \mathsf{Type};\ \ \mathsf{lhs}, \mathsf{rhs} : (e : \mathsf{E}) \longrightarrow \overline{\mathsf{QW}}\ \Sigma\ (\mathsf{Ctx}\ e)\}$$

$$\begin{aligned}
&\mathsf{Inductive}\ \mathsf{QW}\ (\Sigma : \mathsf{Sig})\ (\mathcal{E} : \mathsf{EqTh}\ \Sigma) : \mathsf{Type} := \\
&\quad |\ \mathsf{op}\ (c : \Sigma.\mathsf{Op})\ (f : \Sigma.\mathsf{Ar}\ c \longrightarrow \mathsf{QW}\ \Sigma\ \mathcal{E}) : \mathsf{QW}\ \Sigma\ \mathcal{E} \\
&\quad |\ \mathsf{eq}\ (e : \mathcal{E}.\mathsf{E})\ (\gamma : \mathcal{E}.\mathsf{Ctx}\ e \longrightarrow \mathsf{QW}\ \Sigma\ \mathcal{E}) : (\mathcal{E}.\mathsf{lhs}\ e)\langle\gamma\rangle = (\mathcal{E}.\mathsf{rhs}\ e)\langle\gamma\rangle
\end{aligned}$$

where "substitution" func. $\_\langle\_\rangle : \overline{\mathsf{QW}}\ \Sigma\ \Gamma \longrightarrow (\Gamma \longrightarrow \mathsf{QW}\ \Sigma\ \mathcal{E}) \longrightarrow \mathsf{QW}\ \Sigma\ \mathcal{E}$ defined by

$$(\mathsf{var}\ x)\langle\gamma\rangle := \gamma\ x \qquad (\mathsf{op}\ c\ f)\langle\gamma\rangle := \mathsf{op}\ c\ (\lambda x.(f x)\langle\gamma\rangle)$$

Constructing QITs from QW in OTT

$$\text{QIT scheme} \longrightarrow \text{QW}$$

Constructing QITs from QW in OTT

$$\text{QIT scheme} \longrightarrow \text{QW}$$

Constructing QITs from QW in OTT

$$\text{QIT scheme} \xrightarrow[\color{blue}{\mathrm{B}}]{\hspace{3em}} \mathrm{Tm} \xrightarrow[\color{red}{\mathrm{A}}]{\hspace{3em}} \mathrm{QW}$$

Fiore et al.'s QW Types

$$\mathsf{Sig} = \mathsf{record}\ \{\mathsf{Op} : \mathsf{Type};\ \ \mathsf{Ar} : \mathsf{Op} \longrightarrow \mathsf{Type}\}$$

$$\mathsf{Inductive}\ \overline{\mathsf{QW}}\ (\Sigma : \mathsf{Sig})(\Gamma : \mathsf{Type}) : \mathsf{Type} :=$$

$$\mid \mathsf{var}\ (x : \Gamma) : \overline{\mathsf{QW}}\ \Sigma\ \Gamma$$

$$\mid \mathsf{op}\ (c : \Sigma.\mathsf{Op})\ (f : \Sigma.\mathsf{Ar}\ c \longrightarrow \overline{\mathsf{QW}}\ \Sigma\ \Gamma) : \overline{\mathsf{QW}}\ \Sigma\ \Gamma$$

$$\mathsf{EqTh}\ \Sigma = \mathsf{record}\ \{\mathsf{E} : \mathsf{Type};\ \ \mathsf{Ctx} : \mathsf{E} \longrightarrow \mathsf{Type};\ \ \mathsf{lhs}, \mathsf{rhs} : (e : \mathsf{E}) \longrightarrow \overline{\mathsf{QW}}\ \Sigma\ (\mathsf{Ctx}\ e)\}$$

$$\mathsf{Inductive}\ \mathsf{QW}\ (\Sigma : \mathsf{Sig})\ (\mathcal{E} : \mathsf{EqTh}\ \Sigma) : \mathsf{Type} :=$$

$$\mid \mathsf{op}\ (c : \Sigma.\mathsf{Op})\ (f : \Sigma.\mathsf{Ar}\ c \longrightarrow \mathsf{QW}\ \Sigma\ \mathcal{E}) : \mathsf{QW}\ \Sigma\ \mathcal{E}$$

$$\mid \mathsf{eq}\ (e : \mathcal{E}.\mathsf{E})\ (\gamma : \mathcal{E}.\mathsf{Ctx}\ e \longrightarrow \mathsf{QW}\ \Sigma\ \mathcal{E}) : (\mathcal{E}.\mathsf{lhs}\ e)\langle\gamma\rangle = (\mathcal{E}.\mathsf{rhs}\ e)\langle\gamma\rangle$$

where "substitution" func. $\_\langle\_\rangle : \overline{\mathsf{QW}}\ \Sigma\ \Gamma \longrightarrow (\Gamma \longrightarrow \mathsf{QW}\ \Sigma\ \mathcal{E}) \longrightarrow \mathsf{QW}\ \Sigma\ \mathcal{E}$ defined by

$$(\mathsf{var}\ x)\langle\gamma\rangle := \gamma\ x \qquad (\mathsf{op}\ c\ f)\langle\gamma\rangle := \mathsf{op}\ c\ (\lambda x.(f\,x)\langle\gamma\rangle)$$

Our finitary universal QIT (see infinitary one in the github repo)

$$\mathrm{Sig} = \text{record}\ \{\mathrm{Op} : \mathrm{Type};\ \ \mathsf{Ar} : \mathsf{Op} \longrightarrow \mathsf{Nat}\}$$

$$\text{Inductive } \overline{\mathsf{Tm}}\ (\Sigma : \mathrm{Sig})(\Gamma : \mathrm{Type}) : \mathrm{Type} :=$$

$$\mid \mathrm{var}\ (x : \Gamma) : \overline{\mathsf{Tm}}\ \Sigma\ \Gamma$$

$$\mid \mathrm{op}\ (c : \Sigma.\mathrm{Op})\ (\mathbf{t} : \mathsf{Vec}\ (\overline{\mathsf{Tm}}\ \Sigma\ \Gamma)\ (\Sigma.\mathsf{Ar}\ c)) : \overline{\mathsf{Tm}}\ \Sigma\ \Gamma$$

$$\mathrm{EqTh}\ \Sigma = \text{record}\ \{\mathsf{E} : \mathrm{Type};\ \ \mathsf{Ctx} : \mathsf{E} \longrightarrow \mathrm{Type};\ \ \mathsf{lhs}, \mathsf{rhs} : (e : \mathsf{E}) \longrightarrow \overline{\mathsf{Tm}}\ \Sigma\ (\mathsf{Ctx}\ e)\}$$

$$\text{Inductive } \mathsf{Tm}\ (\Sigma : \mathrm{Sig})\ (\mathcal{E} : \mathrm{EqTh}\ \Sigma) : \mathrm{Type} :=$$

$$\mid \mathrm{op}\ (c : \Sigma.\mathrm{Op})\ (\mathbf{t} : \mathsf{Vec}\ (\mathsf{Tm}\ \Sigma\ \mathcal{E})\ (\Sigma.\mathsf{Ar}\ c)) : \mathsf{Tm}\ \Sigma\ \mathcal{E}$$

$$\mid \mathrm{eq}\ (e : \mathcal{E}.\mathsf{E})\ (\gamma : \mathcal{E}.\mathsf{Ctx}\ e \longrightarrow \mathsf{Tm}\ \Sigma\ \mathcal{E}) : (\mathcal{E}.\mathsf{lhs}\ e)\langle\gamma\rangle = (\mathcal{E}.\mathsf{rhs}\ e)\langle\gamma\rangle$$

where "substitution" func. $\_\langle\_\rangle : \overline{\mathsf{Tm}}\ \Sigma\ \Gamma \longrightarrow (\Gamma \longrightarrow \mathsf{Tm}\ \Sigma\ \mathcal{E}) \longrightarrow \mathsf{Tm}\ \Sigma\ \mathcal{E}$ defined by

$$(\mathrm{var}\ x)\langle\gamma\rangle := \gamma\ x \qquad (\mathrm{op}\ c\ [t_1, \ldots, t_k])\langle\gamma\rangle := \mathrm{op}\ c\ [t_1\langle\gamma\rangle, \ldots, t_k\langle\gamma\rangle]$$

Constructing QITs from QW in OTT

$$\text{QIT scheme} \xrightarrow[\color{blue}{\mathrm{B}}]{} \mathrm{Tm} \xrightarrow[\color{red}{\mathrm{A}}]{} \mathrm{QW}$$

Constructing QITs from QW in OTT

$$\text{QIT scheme} \xrightarrow[\color{blue}{\text{B}}]{} \text{Tm} \xrightarrow[\color{red}{\text{A}}]{} \text{QW}$$

Construction A (**)** In OTT (with $\mathrm{cast}_p^{A \rightsquigarrow A}\, t \equiv t$) we can construct Tm from QW.

Constructing QITs from QW in OTT

$$\text{QIT scheme} \xrightarrow[\text{B}]{} \text{Tm} \xrightarrow[\text{A}]{} \text{QW}$$

Construction A In OTT (with $\mathrm{cast}_p^{A \rightsquigarrow A} t \equiv t$) we can construct Tm from QW.

Proof (tedious) Involves switching between first- and higher-order representation of branching.

Constructing QITs from QW in OTT

$$\text{QIT scheme} \xrightarrow[\text{B}]{} \text{Tm} \xrightarrow[\text{A}]{} \text{QW}$$

Construction A ( ) In OTT (with $\text{cast}_p^{A \rightsquigarrow A} t \equiv t$) we can construct Tm from QW.

Proof (tedious) Involves switching between first- and higher-order representation of branching.

Construction B (WIP) Non-indexed infinitary QITs can be constructed from Tm.

Constructing QITs from QW in OTT

$$\text{QIT scheme} \xrightarrow[\text{B}]{} \text{Tm} \xrightarrow[\text{A}]{} \text{QW}$$

Construction A () In OTT (with $\mathrm{cast}_p^{A \rightsquigarrow A}\, t \equiv t$) we can construct Tm from QW.

Proof (tedious) Involves switching between first- and higher-order representation of branching.

Construction B (WIP) Non-indexed infinitary QITs can be constructed from Tm.

Proof Not yet written, but examples suggest it is direct (see github in abstract).

Example: MSet

```
Definition MSet_Arity@{i} (A : Type@{i}) : Sum Unit A -> MList@{Set+1} Type@{Set} := Sum_rect _ _ _ (fun _ => mnil) (fun _ => mskip mnil).

Definition MSet_Sig@{i} (A : Type@{i}) : Sig@{i Set} := {| C := Sum Unit A; Arity := MSet_Arity A |}.

Definition MSet0@{i} (A : Type@{i}) (Γ : Type@{Set}): Type@{i} := Tm0 (MSet_Sig A) Γ.

Definition nil0@{i} {A} {Γ} : MSet0@{i} A Γ := @sym0 (MSet_Sig A) Γ (inl I) I.

Definition cons0@{i} {A} {Γ} (a : A) (m : MSet0 A Γ) : MSet0@{i} A Γ := @sym0 (MSet_Sig A) _ (inr a) {| fst := m; snd := I|}.

Definition MSet_lhs@{i} A (e : A × A) : MSet0@{i} A Unit := cons0 (fst e) (cons0 (snd e) (var0' I)).

Definition MSet_rhs@{i} A (e : A × A) : MSet0@{i} A Unit := cons0 (snd e) (cons0 (fst e) (var0' I)).

Definition MSet_EqTh@{i} (A : Type@{i}) : EqTh (MSet0 A) := {| E := A × A; Ctx := fun _ => Unit; lhs := MSet_lhs@{i} A; rhs := MSet_rhs@{i} A |}.

Definition MSet@{i} (A : Type@{i}) : Type@{i} := Tm (MSet_Sig A) (MSet_EqTh A).

Definition nil'@{i} {A} : MSet@{i} A := @sym (MSet_Sig A) _ (inl I) I.

Definition cons'@{i} {A} (a : A) (m : MSet A) : MSet@{i} A := @sym (MSet_Sig A) _ (inr a) {| fst := m; snd := I|}.

Definition MSet_eq@{i} {A} (x y : A) (m : MSet@{i} A) : cons' x (cons' y m) ~ cons' y (cons' x m)
    := @eq _ (MSet_EqTh A) {| fst := x; snd := y|} (fun e => m).

Definition MSet_rect@{i j} {A} (ϵMSet : MSet A -> Type@{j}) (ϵnil  : ϵMSet nil') (ϵcons : forall a m, ϵMSet m -> ϵMSet (cons' a m))
    (ϵeq : forall x y m (ϵm : ϵMSet m), let p := (obseq_sym (ap ϵMSet (MSet_eq@{i} x y m)) in ϵcons x _ (ϵcons y _ ϵm) ~ (p # ϵcons y _ (ϵcons x _ ϵm)))
    (x : MSet A) : ϵMSet x.
    refine (recTm {| ϵTm := ϵMSet ; ϵsym := _; ϵeq := _|} x). shelve. Unshelve.
    - intros. destruct c.
      + destruct u. destruct l. apply ϵnil.
      + apply ϵcons. apply (fst ϵl).
    - intros. apply ϵeq.
Defined.
```

Conclusion

A construction of infinitary non-indexed QITs with definitional β-rules in OTT+QW.

Conclusion

A construction of infinitary non-indexed QITs with definitional β-rules in OTT+QW.

Next steps Establish the metatheory of OTT+QW:

Conclusion

A construction of infinitary non-indexed QITs with definitional β-rules in OTT+QW.

Next steps Establish the metatheory of OTT+QW:

1. Normalization and decidability of conversion:
   Logical relations, or by simulation with OTT+W?

Conclusion

A construction of infinitary non-indexed QITs with definitional β-rules in OTT+QW.

Next steps Establish the metatheory of OTT+QW:

1. Normalization and decidability of conversion:
   Logical relations, or by simulation with OTT+W?
2. Consistency: Adapting set-theoretic model of Pujet and Tabareau.

Conclusion

A construction of infinitary non-indexed QITs with definitional β-rules in OTT+QW.

Next steps Establish the metatheory of OTT+QW:

1. Normalization and decidability of conversion:
   Logical relations, or by simulation with OTT+W?
2. Consistency: Adapting set-theoretic model of Pujet and Tabareau.

A restricted form of canonicity for the constructed QITs should follow.

Conclusion

A construction of infinitary non-indexed QITs with definitional β-rules in OTT+QW.

Next steps Establish the metatheory of OTT+QW:

1. Normalization and decidability of conversion:
   Logical relations, or by simulation with OTT+W?
2. Consistency: Adapting set-theoretic model of Pujet and Tabareau.

A restricted form of canonicity for the constructed QITs should follow.

Future work

Conclusion

A construction of infinitary non-indexed QITs with definitional β-rules in OTT+QW.

Next steps Establish the metatheory of OTT+QW:

1. Normalization and decidability of conversion:
   Logical relations, or by simulation with OTT+W?
2. Consistency: Adapting set-theoretic model of Pujet and Tabareau.

A restricted form of canonicity for the constructed QITs should follow.

Future work

- Add primitive scheme of QITs to observational version of Rocq.

Conclusion

A construction of infinitary non-indexed QITs with definitional β-rules in OTT+QW.

Next steps Establish the metatheory of OTT+QW:

1. Normalization and decidability of conversion:
   Logical relations, or by simulation with OTT+W?
2. Consistency: Adapting set-theoretic model of Pujet and Tabareau.

A restricted form of canonicity for the constructed QITs should follow.

Future work

- Add primitive scheme of QITs to observational version of Rocq.
- Explore more complex classes of types (indexed QITs and QIITs).

Conclusion

A construction of infinitary non-indexed QITs with definitional β-rules in OTT+QW.

Next steps Establish the metatheory of OTT+QW:

1. Normalization and decidability of conversion:
   Logical relations, or by simulation with OTT+W?
2. Consistency: Adapting set-theoretic model of Pujet and Tabareau.

A restricted form of canonicity for the constructed QITs should follow.

Future work

- Add primitive scheme of QITs to observational version of Rocq.
- Explore more complex classes of types (indexed QITs and QIITs).

Thank you for your attention!